Setting up Certifying Authority using OpenSSL

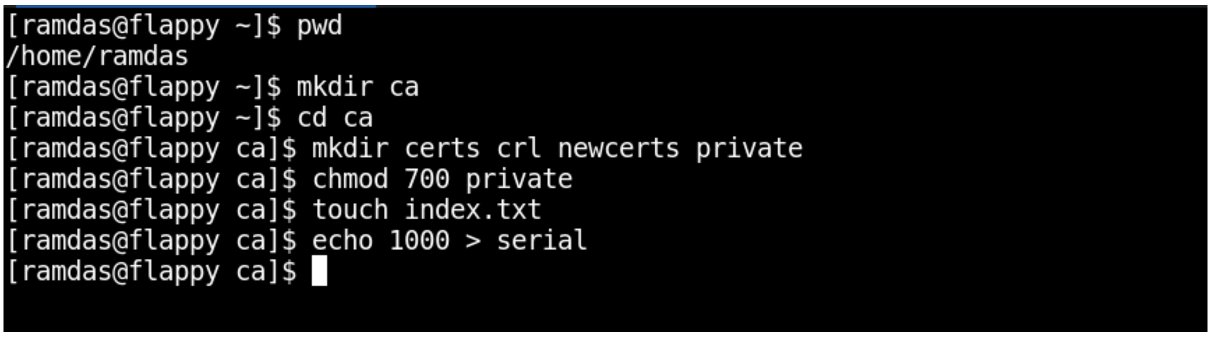
In this assignment we are going to setup a toy certifying authority to issue **x509** digital certificates. It will comprise of following steps.

1. Set up a Root Certifying Authority (**GBSRootCA**). Create a public key-private key pair for GBSRootCA and create a self-signed digital certificate for GBSRootCA.
2. Set up an Intermediate Certifying Authority (**GBSCSTCA**). Create a public key-private key pair for GBSCSTCA and get a digital certificate issued for it signed by GBSRootCA.
3. Create a public–private key pair for website **www.gbs.unigoa.ac.in** and get is public key certified by GBSCSTCA.
4. Demonstrate how certificates can be verified.
5. Create certificates for individual users and show how to revoke certificates. Create a Certificate Revocation List (CRL).
6. Understand how we can use Online Certificate Status Protocol (OCSP) to determine revocation status of a certificate over the network.

All the commands shown below have been tested on Fedora version 25. You are encouraged to read the man pages of all the commands.

Create the required directory structure

In your home directory do the following.



```
[ramdas@flappy ~]$ pwd
/home/ramdas
[ramdas@flappy ~]$ mkdir ca
[ramdas@flappy ~]$ cd ca
[ramdas@flappy ca]$ mkdir certs crl newcerts private
[ramdas@flappy ca]$ chmod 700 private
[ramdas@flappy ca]$ touch index.txt
[ramdas@flappy ca]$ echo 1000 > serial
[ramdas@flappy ca]$ 
```

The **index.txt** and **serial** files act as a flat file database to keep track of signed certificates and their serial numbers.

Create configuration file

Operations performed to create/sign certificates require the private key of the CA to be specified. Also different certificates will be created with different extensions based upon the purpose for which they will be used. Instead of passing this information at the command prompt every time the various commands are invoked, we will create a configuration file to store this information. This configuration file will have different sections to store the relevant information. The commands will pick up the information they require from the respective section.

We will need two separate configuration files, one for Root CA and the other for Intermediate CA. Let's begin by creating the configuration file for Root CA. The file **openssl.cnf** will contain all the configuration related information required for creating key pairs, certifications signing requests and the x509 digital certificates. The file can be created using any text editor and should contain following information. The file should be located in **ca directory.** The file contains multiple sections required for different purposes. Highlighted parts are comments. Avoid any spelling mistakes.

```
[ ca ]
#The [ ca ] section is mandatory. We tell OpenSSL to use the options from
the [ CA_default ] section for CA related information.
default_ca = CA_default

[ CA_default ]
# Directory and file locations. Remember to change the dir value to point to your home
directory.
dir                  = /home/ramdas/ca
certs                 = $dir/certs
crl_dir              = $dir/crl
new_certs_dir     = $dir/newcerts
database           = $dir/index.txt
serial               = $dir/serial
RANDFILE          = $dir/private/.rand

# The root key and root certificate.
private_key      = $dir/private/GBSRootCA.key
certificate       = $dir/certs/GBSRootCA.cert

# For certificate revocation lists.
crlnumber          = $dir/crlnumber
crl                   = $dir/crl/GBSRootCA.crl
crl_extensions     = crl_ext
default_crl_days   = 30

# SHA-1 is deprecated, so use SHA-2 hash instead for signing purpose.
default_md        = sha256

name_opt         = ca_default
cert_opt          = ca_default
```

```
default_days      = 375
preserve          = no
policy            = policy_strict

[ policy_strict ]
# The root CA should only sign intermediate certificates that match.
countryName               = match
stateOrProvinceName       = match
organizationName          = match
organizationalUnitName    = optional
commonName                = supplied
emailAddress              = optional

[ policy_loose ]
# Allow the intermediate CA to sign a more diverse range of certificates.
countryName               = optional
stateOrProvinceName       = optional
localityName              = optional
organizationName          = optional
organizationalUnitName    = optional
commonName                = supplied
emailAddress              = optional

[ req ]
# Options for the openssl req command
default_bits        = 2048
distinguished_name  = req_distinguished_name
string_mask         = utf8only

# SHA-1 is deprecated, so use SHA-2 hash instead.
default_md          = sha256

# Extension to add when the -x509 option is used.
x509_extensions     = v3_ca

[ req_distinguished_name ]
#This section declares the information normally required in a certificate signing request.
countryName                     = Country Name (2 letter code)
stateOrProvinceName             = State or Province Name
localityName                    = Locality Name
0.organizationName              = Organization Name
```

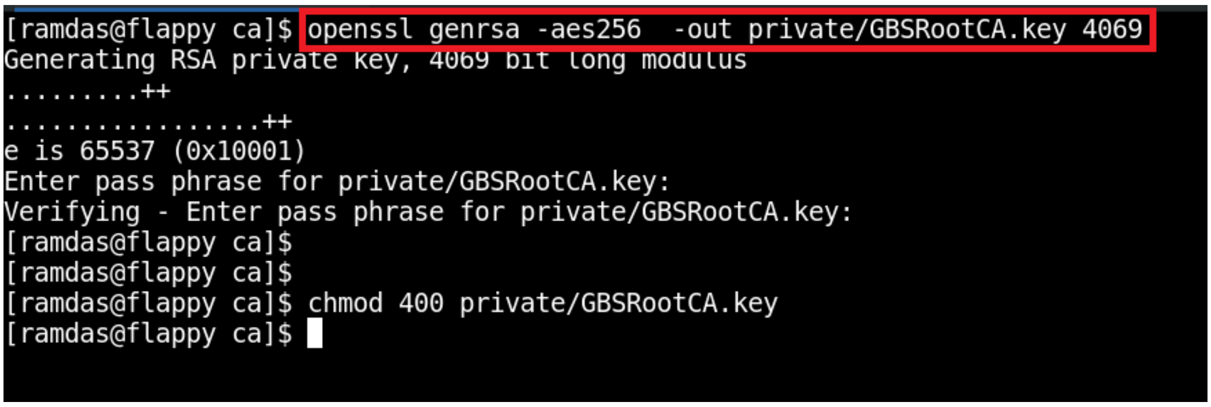
```
organizationalUnitName      = Organizational Unit Name
commonName                  = Common Name
emailAddress                = Email Address

# Optionally, specify some defaults.
countryName_default              = IN
stateOrProvinceName_default      = Goa
localityName_default             = Taleigao
0.organizationName_default       = Goa University

[ v3_ca ]
# Extensions for a typical CA. This extension is used when we create the certificate of Root
CA.
subjectKeyIdentifier    = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints        = critical, CA:true
keyUsage                = critical, digitalSignature, cRLSign, keyCertSign

[ v3_intermediate_ca ]
# Extensions for a typical intermediate CA. This extension used when we create the
intermediate CA certificate.
#pathlen:0 ensures that there can be no further certificate authorities below the
intermediate CA.
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints        = critical, CA:true, pathlen:0
keyUsage                = critical, digitalSignature, cRLSign, keyCertSign

[ usr_cert ]
# Extensions for client certificates.
basicConstraints        = CA:FALSE
nsCertType              = client, email
nsComment               = "OpenSSL Generated Client Certificate"
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid,issuer
keyUsage                = critical, nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage       = clientAuth, emailProtection

[ server_cert ]
# Extensions for server certificates.
basicConstraints         = CA:FALSE
```

```
nsCertType              = server
nsComment               = "OpenSSL Generated Server Certificate"
subjectKeyIdentifier    = hash
authorityKeyIdentifier = keyid,issuer:always
keyUsage                = critical, digitalSignature, keyEncipherment
extendedKeyUsage        = serverAuth

[ crl_ext ]
# Extension for CRLs.
authorityKeyIdentifier  =keyid:always

[ ocsp ]
# Extension for OCSP signing certificates.
basicConstraints        = CA:FALSE
subjectKeyIdentifier    = hash
authorityKeyIdentifier = keyid,issuer
keyUsage                = critical, digitalSignature
extendedKeyUsage        = critical, OCSPSigning
```

Setting up Root CA

We begin by generating 4096 bit RSA public-private key pair for Root CA. The private key is protected using 256 bit AES encryption **(-aes256 option)**. The 256 bit key required for AES is generated from the password supplied at standard input. Any time we need to use the private key, the password will have to be supplied.

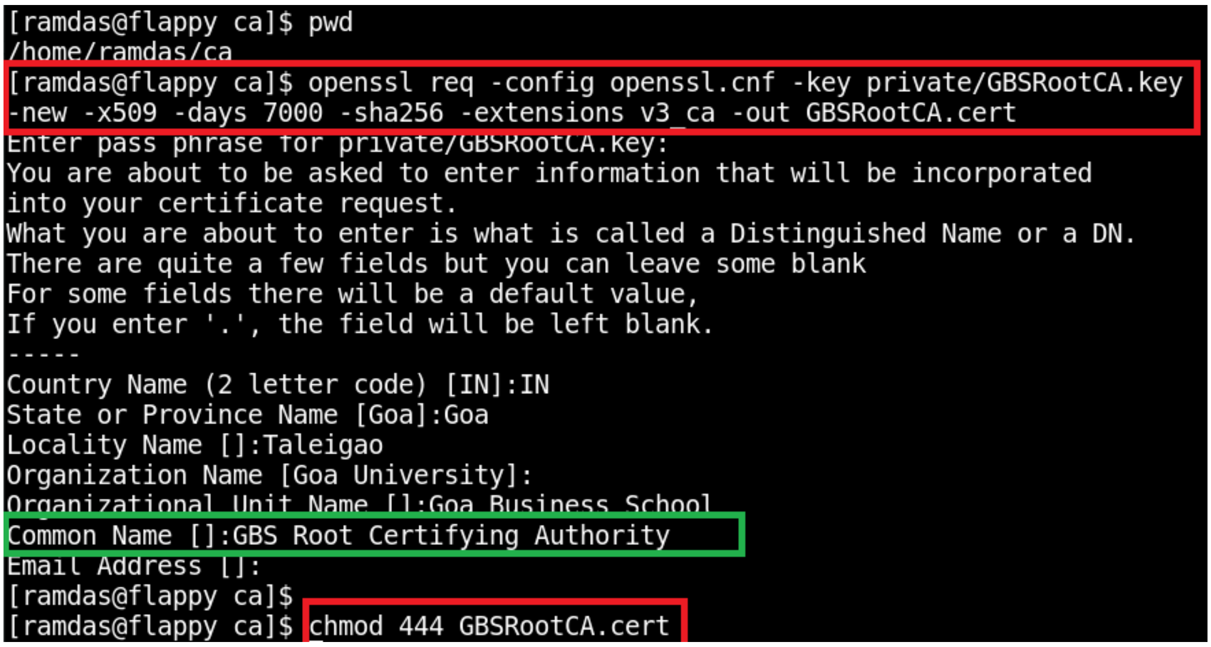
The key file (**GBSRootCA.key**) is stored in **private subdirectory** and permissions are changed so that only the owner can read the file.



```
[ramdas@flappy ca]$ openssl genrsa -aes256  -out private/GBSRootCA.key 4069
Generating RSA private key, 4069 bit long modulus
.........++
.................++
e is 65537 (0x10001)
Enter pass phrase for private/GBSRootCA.key:
Verifying - Enter pass phrase for private/GBSRootCA.key:
[ramdas@flappy ca]$
[ramdas@flappy ca]$
[ramdas@flappy ca]$ chmod 400 private/GBSRootCA.key
[ramdas@flappy ca]$
```

Now we will create the Root CA digital certificate. Remember that the Root CA digital certificate is always self-signed. The issuer name is same as the subject name in such certificate. This is necessary as in the chain of trust there is nobody above Root CA. The Root CA authenticates itself.

This will require two steps. First we create a new Certificate Signing Request CSR **(-new option**) and then create the self-signed certificate (**-x509 option**). We will use single invocation of **openssl req** command to simultaneously perform both the steps. We will use **section v3_ca (-extensions option**) from configuration file (**-config option**) for the purpose of creating this certificate. This section states that the certificate created is a CA certificate (**basicConstraints = critical, CA:true**),. That is, it will be used for the purpose of signing digital certificates and the CRLs (**keyUsage = critical, digitalSignature, cRLSign, keyCertSign**). That is, the private key of Root CA will be used for signing digital certificates and the self-signed digital certificate which contains the public key of Root CA can be used for verification. We use **–key** option to pass the private key and **–days** option to specify the period for which the certificate will be valid.

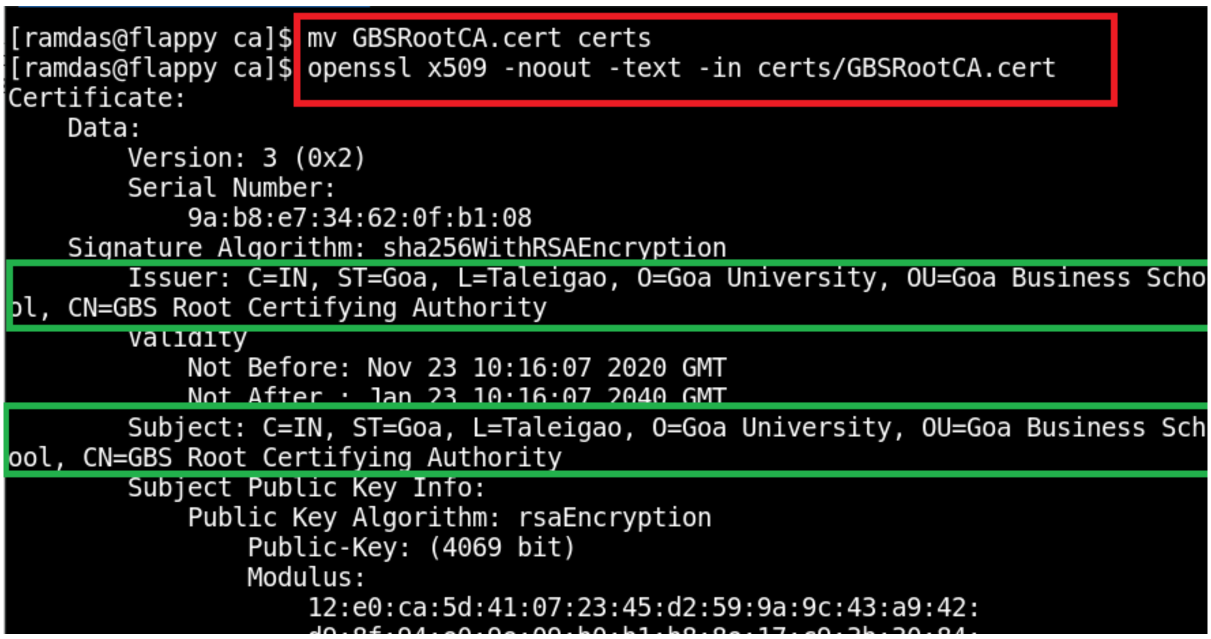


```
[ramdas@flappy ca]$ pwd
/home/ramdas/ca
[ramdas@flappy ca]$ openssl req -config openssl.cnf -key private/GBSRootCA.key
-new -x509 -days 7000 -sha256 -extensions v3_ca -out GBSRootCA.cert
Enter pass phrase for private/GBSRootCA.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [IN]:IN
State or Province Name [Goa]:Goa
Locality Name []:Taleigao
Organization Name [Goa University]:
Organizational Unit Name []:Goa Business School
Common Name []:GBS Root Certifying Authority
Email Address []:
[ramdas@flappy ca]$
[ramdas@flappy ca]$ chmod 444 GBSRootCA.cert
```

Move the generated certificate to **certs subdirectory**. We can check the contents of created certificate using the **openssl x509** command.

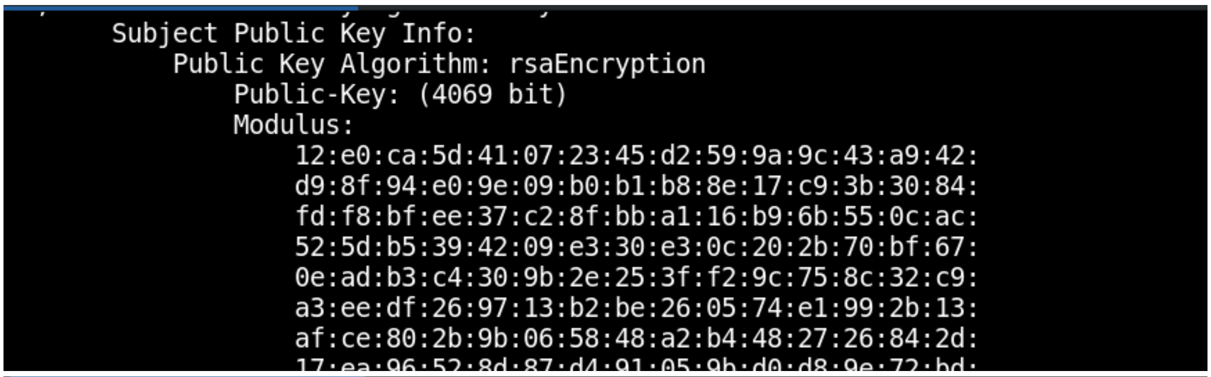
-noout option says don’t output the raw contents of digital certificate file. This is the default action.

-text option says output the fields of the digital certificate properly formatted.

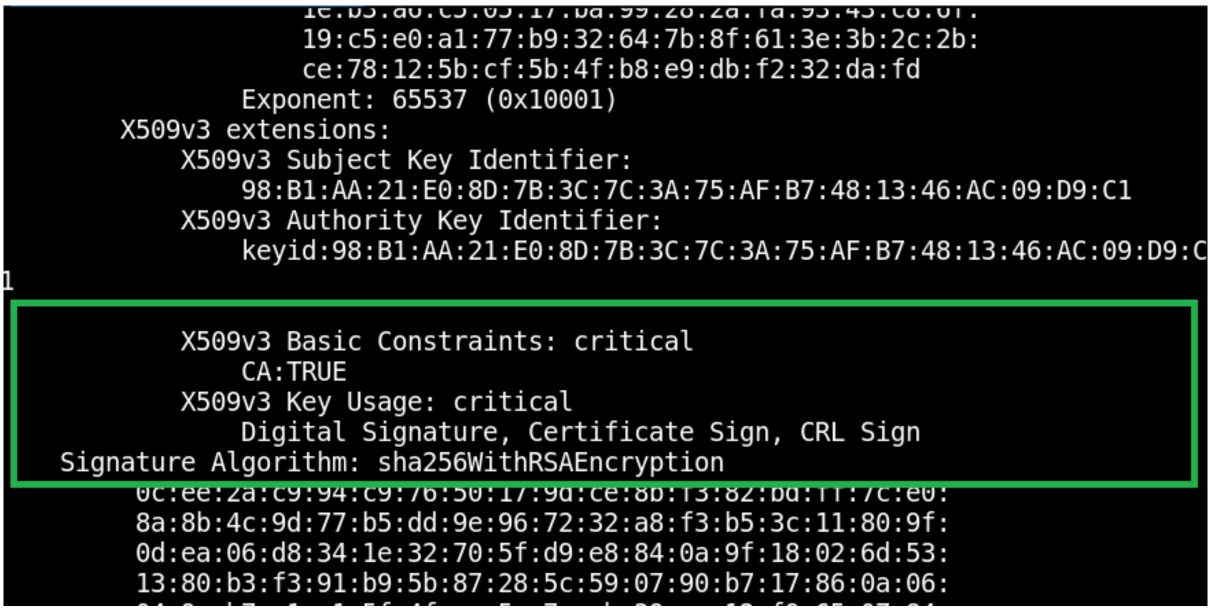


```
[ramdas@flappy ca]$ mv GBSRootCA.cert certs
[ramdas@flappy ca]$ openssl x509 -noout -text -in certs/GBSRootCA.cert
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            9a:b8:e7:34:62:0f:b1:08
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=IN, ST=Goa, L=Taleigao, O=Goa University, OU=Goa Business Scho
ol, CN=GBS Root Certifying Authority
        Validity
            Not Before: Nov 23 10:16:07 2020 GMT
            Not After : Jan 23 10:16:07 2040 GMT
        Subject: C=IN, ST=Goa, L=Taleigao, O=Goa University, OU=Goa Business Sch
ool, CN=GBS Root Certifying Authority
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4069 bit)
                Modulus:
                    12:e0:ca:5d:41:07:23:45:d2:59:9a:9c:43:a9:42:
```

We can very clearly see that issuer and the subject is the same. This is a self-signed certificate. For signing purpose the SHA256 bit hash of the certificate contents have been encrypted using the private key of the Root CA.



```
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4069 bit)
                Modulus:
                    12:e0:ca:5d:41:07:23:45:d2:59:9a:9c:43:a9:42:
                    d9:8f:94:e0:9e:09:b0:b1:b8:8e:17:c9:3b:30:84:
                    fd:f8:bf:ee:37:c2:8f:bb:a1:16:b9:6b:55:0c:ac:
                    52:5d:b5:39:42:09:e3:30:e3:0c:20:2b:70:bf:67:
                    0e:ad:b3:c4:30:9b:2e:25:3f:f2:9c:75:8c:32:c9:
                    a3:ee:df:26:97:13:b2:be:26:05:74:e1:99:2b:13:
                    af:ce:80:2b:9b:06:58:48:a2:b4:48:27:26:84:2d:
                    17:ea:96:52:8d:87:d4:91:05:9b:d0:d8:9e:72:bd:
```



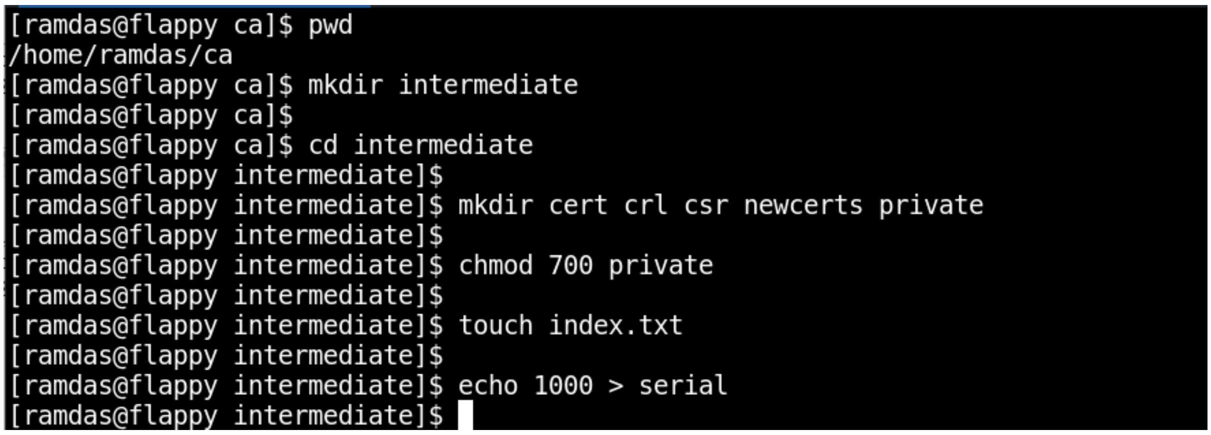
```
                    1e:b3:a6:c5:05:17:ba:99:28:2a:1a:93:43:c8:6f:
                    19:c5:e0:a1:77:b9:32:64:7b:8f:61:3e:3b:2c:2b:
                    ce:78:12:5b:cf:5b:4f:b8:e9:db:f2:32:da:fd
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                98:B1:AA:21:E0:8D:7B:3C:7C:3A:75:AF:B7:48:13:46:AC:09:D9:C1
            X509v3 Authority Key Identifier:
                keyid:98:B1:AA:21:E0:8D:7B:3C:7C:3A:75:AF:B7:48:13:46:AC:09:D9:C
1

            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
    Signature Algorithm: sha256WithRSAEncryption
         0c:ee:2a:c9:94:c9:76:50:17:9d:ce:8b:f3:82:bd:ff:7c:e0:
         8a:8b:4c:9d:77:b5:dd:9e:96:72:32:a8:f3:b5:3c:11:80:9f:
         0d:ea:06:d8:34:1e:32:70:5f:d9:e8:84:0a:9f:18:02:6d:53:
         13:80:b3:f3:91:b9:5b:87:28:5c:59:07:90:b7:17:86:0a:06:
```

In the extension section above, we can see that CA flag has been set to true and the key usage information is also present. This has been picked up from the **v3_ca section** of the configuration file. Both the extensions are marked as critical. This implies that these extensions must be strictly respected.

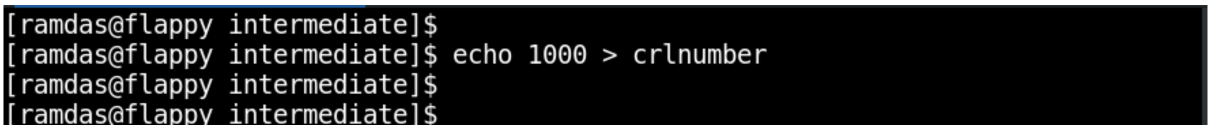
Create the certificate for the intermediate CA

The Intermediate CA is the one which will issue the digital certificates to the end users. The digital certificate of the Intermediate CA will be in turn signed by the Root CA. For the purpose of verifying the end user digital certificate, both the digital certificate of Intermediate CA and the Root CA will be required.

First we will create a subdirectory **intermediate** under directory **ca** to store all the Intermediate CA files.


```
[ramdas@flappy ca]$ pwd
/home/ramdas/ca
[ramdas@flappy ca]$ mkdir intermediate
[ramdas@flappy ca]$
[ramdas@flappy ca]$ cd intermediate
[ramdas@flappy intermediate]$
[ramdas@flappy intermediate]$ mkdir cert crl csr newcerts private
[ramdas@flappy intermediate]$
[ramdas@flappy intermediate]$ chmod 700 private
[ramdas@flappy intermediate]$
[ramdas@flappy intermediate]$ touch index.txt
[ramdas@flappy intermediate]$
[ramdas@flappy intermediate]$ echo 1000 > serial
[ramdas@flappy intermediate]$
```

The file **serial** is required to store the serial number of next certificate to be generated. Similarly the file **crlnumber** is used to keep track of Certificate Revocation List.


```
[ramdas@flappy intermediate]$
[ramdas@flappy intermediate]$ echo 1000 > crlnumber
[ramdas@flappy intermediate]$
[ramdas@flappy intermediate]$
```

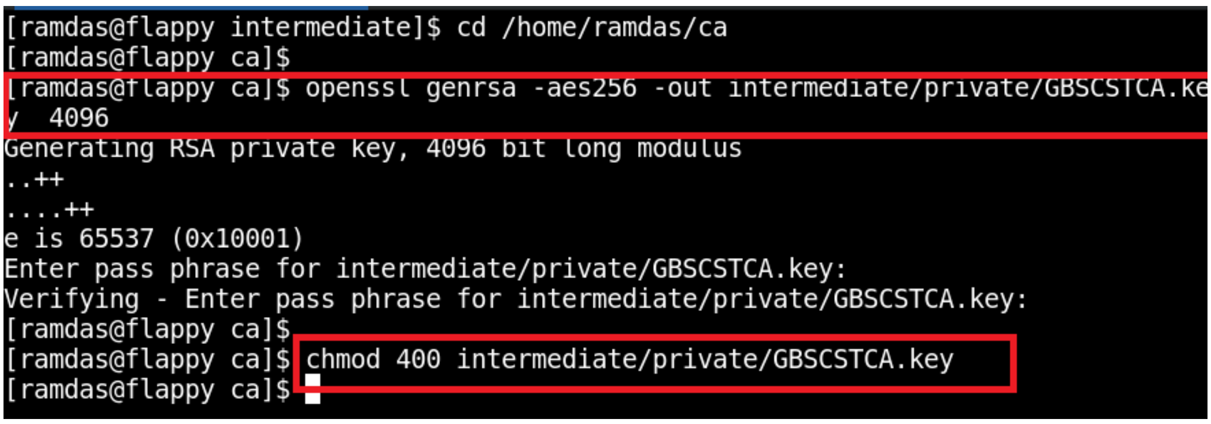
Copy the **openssl.cnf** file from the parent **ca directory** to the **subdirectory intermediate** and change following entries in **CA_default section**. Please remember to point **dir** to your home directory.

```
dir             = /home/ramdas/ca/intermediate
certs           = $dir/cert
private_key     = $dir/private/GBSCSTCA.key
```

```
certificate     = $dir/cert/GBSCSTCA.cert
crl             = $dir/crl/GBSCSTCA.crl

policy          = policy_loose
```

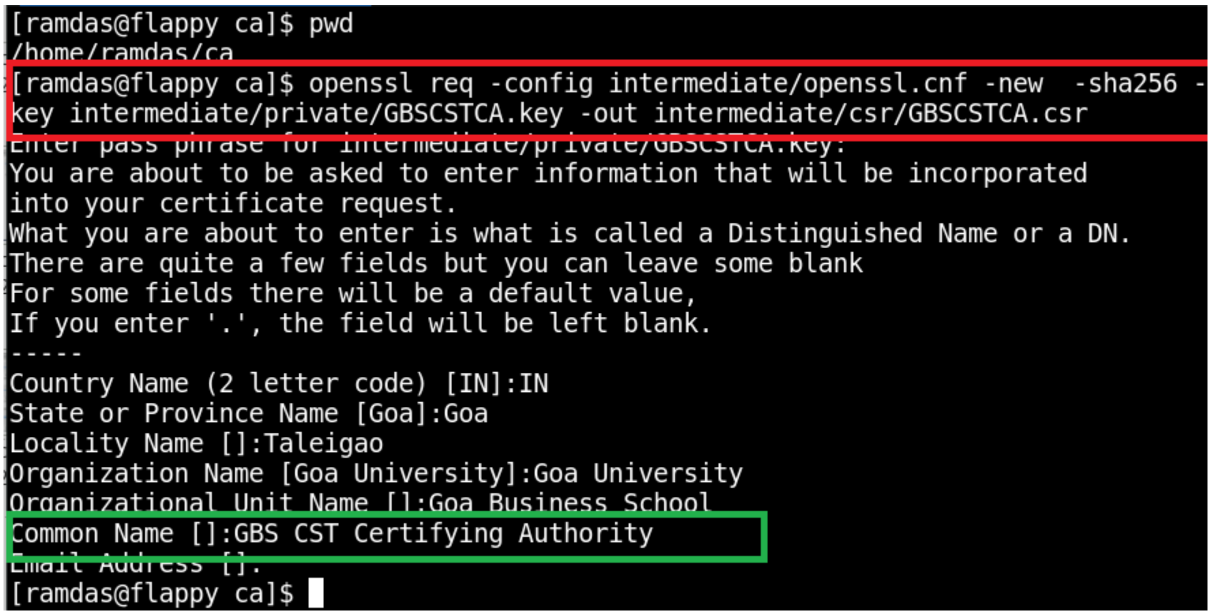
Please observe that the **certs** directory where all the issued certificates are stored is $dir/cert and not $dir/certs as in case of Root CA.

We begin by creating a RSA public-private key pair for the Intermediate CA.



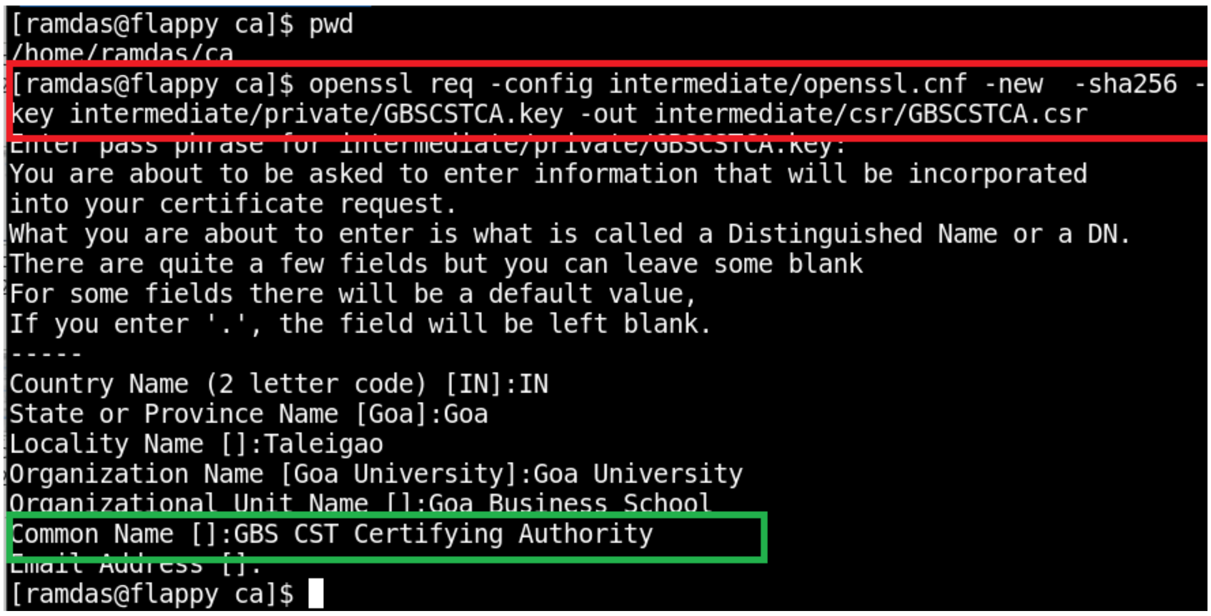
```
[ramdas@flappy intermediate]$ cd /home/ramdas/ca
[ramdas@flappy ca]$
[ramdas@flappy ca]$ openssl genrsa -aes256 -out intermediate/private/GBSCSTCA.ke
y  4096
Generating RSA private key, 4096 bit long modulus
..++
....++
e is 65537 (0x10001)
Enter pass phrase for intermediate/private/GBSCSTCA.key:
Verifying - Enter pass phrase for intermediate/private/GBSCSTCA.key:
[ramdas@flappy ca]$
[ramdas@flappy ca]$ chmod 400 intermediate/private/GBSCSTCA.key
[ramdas@flappy ca]$
```

Next we create the Intermediate CA certificate. Unlike in the previous case this will be a proper two-step process. First we will create a Certificate Signing Request (CSR) using the public key of Intermediate CA and then forward it to Root CA for signing. The CSR will be created using **openssl req** command. The Root CA will sign the request and create the certificate using the **openssl ca** command.



```
[ramdas@flappy ca]$ pwd
/home/ramdas/ca
[ramdas@flappy ca]$ openssl req -config intermediate/openssl.cnf -new  -sha256 -
key intermediate/private/GBSCSTCA.key -out intermediate/csr/GBSCSTCA.csr
Enter pass phrase for intermediate/private/GBSCSTCA.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [IN]:IN
State or Province Name [Goa]:Goa
Locality Name []:Taleigao
Organization Name [Goa University]:Goa University
Organizational Unit Name []:Goa Business School
Common Name []:GBS CST Certifying Authority
Email Address []:
[ramdas@flappy ca]$
```

We use **–new** option to specify that a new CSR request is to be created using the supplied key (**-key option**). The key is located in the **intermediate/private subdirectory**. The CSR is stored in **intermediate/csr** subdirectory. Remember to use the **openssl.cnf** file from **intermediate subdirectory** during CSR creation.



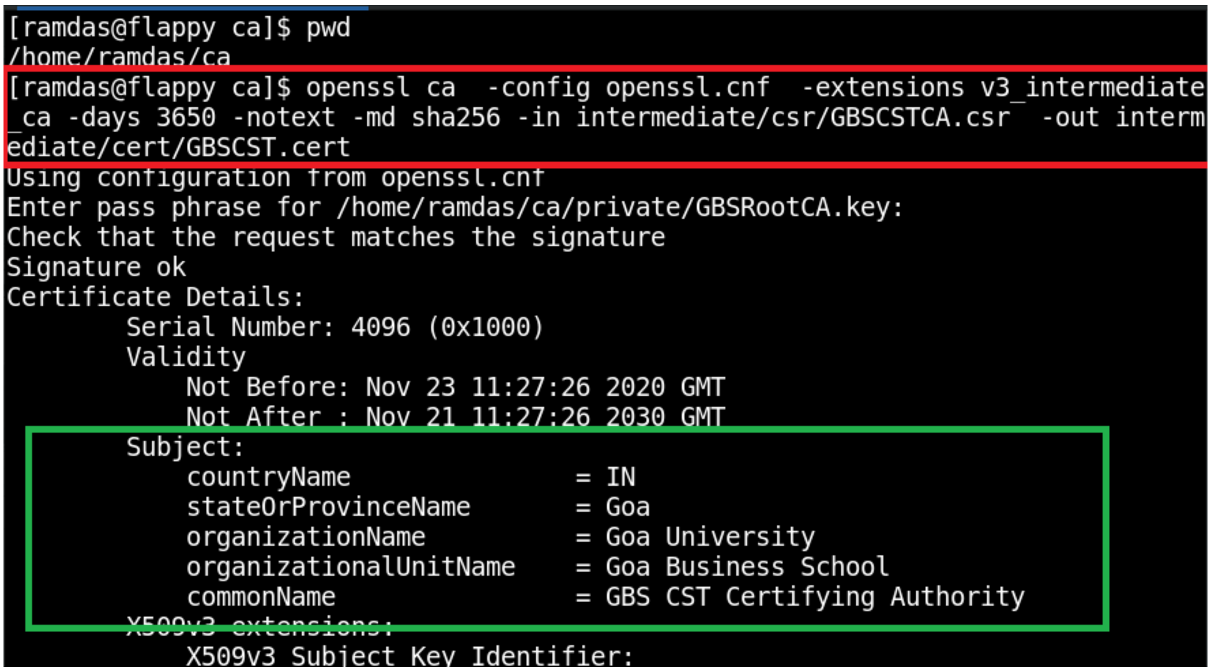
```
[ramdas@flappy ca]$ pwd
/home/ramdas/ca
[ramdas@flappy ca]$ openssl req -config intermediate/openssl.cnf -new  -sha256 -
key intermediate/private/GBSCSTCA.key -out intermediate/csr/GBSCSTCA.csr
Enter pass phrase for intermediate/private/GBSCSTCA.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [IN]:IN
State or Province Name [Goa]:Goa
Locality Name []:Taleigao
Organization Name [Goa University]:Goa University
Organizational Unit Name []:Goa Business School
Common Name []:GBS CST Certifying Authority
Email Address []:
[ramdas@flappy ca]$
```

CSR contains all the information required to generate the digital certificate. Now we can forward the CSR to Root CA to sign and create the Intermediate CA certificate.

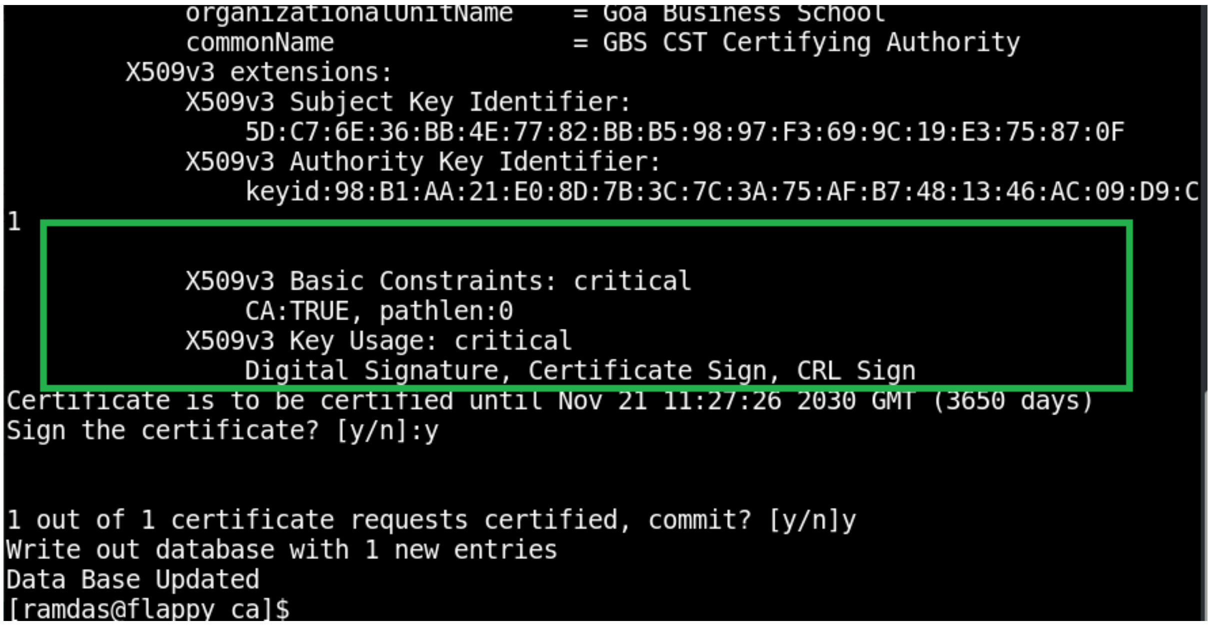
The Root CA configuration file is used for this purpose. The information about where the private key of Root CA is located is in the **CA_ default section** of this configuration file. For the purpose of signing the certificate **v3_inermediate_ca section** is used.

This again identifies that the certificate is a CA certificate and that it can only be used for the purpose signing digital certificate and CRLs. It also sets the **pathlen** extension to zero. This implies the certificate can only be used for the purpose of signing end user certificates and not to sign certificates of another Intermediate CAs as in the case of Root CA. Extension **pathlen** set to zero implies no more additional Intermediate CAs can be present between this CA and the digital certificate of the end user.

The **openssl ca** command invoked below outputs the digital certificate to **GBSCST.cert** file. As per our configuration file the name needs to be **GBSCSTCA.cert**. Please make the correction while invoking the **openssl ca** command.

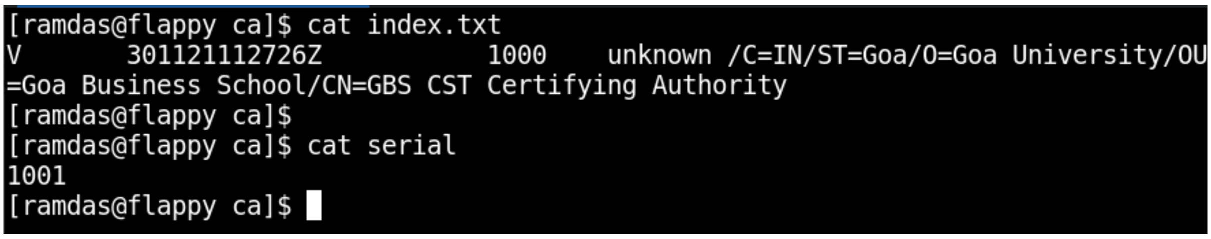


```
[ramdas@flappy ca]$ pwd
/home/ramdas/ca
[ramdas@flappy ca]$ openssl ca  -config openssl.cnf  -extensions v3_intermediate
_ca -days 3650 -notext -md sha256 -in intermediate/csr/GBSCSTCA.csr  -out interm
ediate/cert/GBSCST.cert
Using configuration from openssl.cnf
Enter pass phrase for /home/ramdas/ca/private/GBSRootCA.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4096 (0x1000)
        Validity
            Not Before: Nov 23 11:27:26 2020 GMT
            Not After : Nov 21 11:27:26 2030 GMT
        Subject:
            countryName               = IN
            stateOrProvinceName       = Goa
            organizationName          = Goa University
            organizationalUnitName    = Goa Business School
            commonName                = GBS CST Certifying Authority
        X509v3 extensions:
            X509v3 Subject Key Identifier:
```



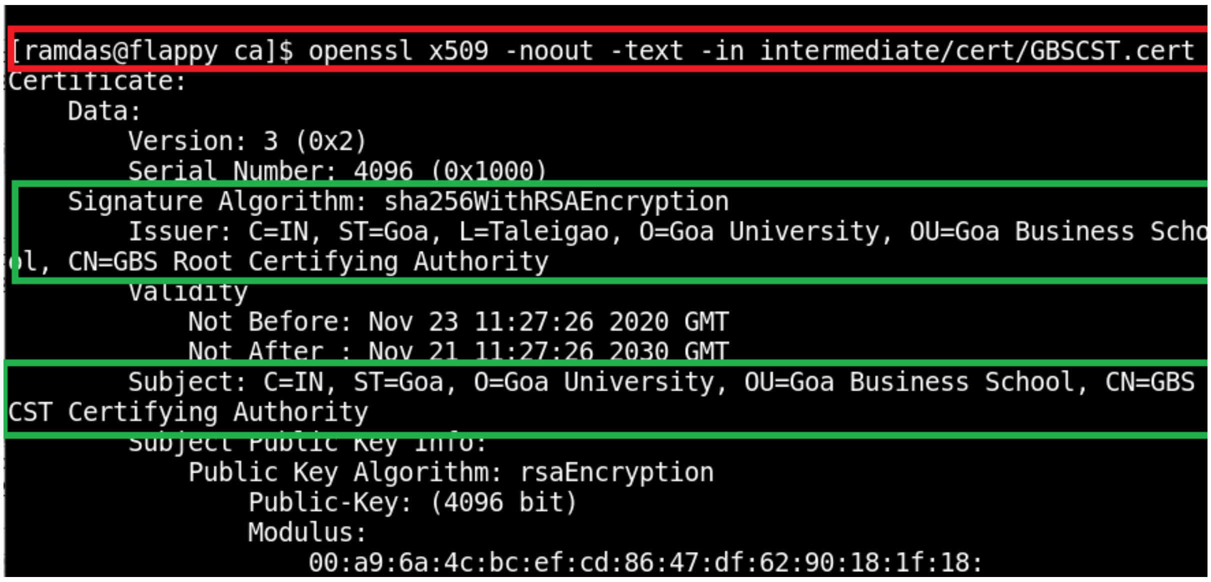
```
            organizationalUnitName    = Goa Business School
            commonName                = GBS CST Certifying Authority
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                5D:C7:6E:36:BB:4E:77:82:BB:B5:98:97:F3:69:9C:19:E3:75:87:0F
            X509v3 Authority Key Identifier:
                keyid:98:B1:AA:21:E0:8D:7B:3C:7C:3A:75:AF:B7:48:13:46:AC:09:D9:C
1

            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
Certificate is to be certified until Nov 21 11:27:26 2030 GMT (3650 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
[ramdas@flappy ca]$
```

The file **index.txt** of Root CA now contains an entry for this certificate and the file **serial** now contains the next serial number, i.e. 1001.



```
[ramdas@flappy ca]$ cat index.txt
V       301121112726Z           1000    unknown /C=IN/ST=Goa/O=Goa University/OU
=Goa Business School/CN=GBS CST Certifying Authority
[ramdas@flappy ca]$
[ramdas@flappy ca]$ cat serial
1001
[ramdas@flappy ca]$ 
```

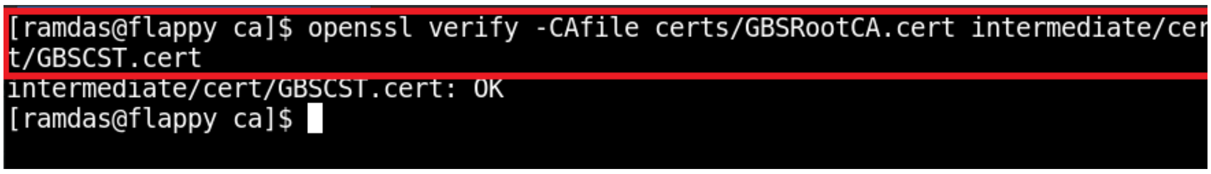
We can look at the contents of the digital certificate by using **openssl x509** command.



```
[ramdas@flappy ca]$ openssl x509 -noout -text -in intermediate/cert/GBSCST.cert
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4096 (0x1000)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=IN, ST=Goa, L=Taleigao, O=Goa University, OU=Goa Business Scho
ol, CN=GBS Root Certifying Authority
        Validity
            Not Before: Nov 23 11:27:26 2020 GMT
            Not After : Nov 21 11:27:26 2030 GMT
        Subject: C=IN, ST=Goa, O=Goa University, OU=Goa Business School, CN=GBS
CST Certifying Authority
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
                    00:a9:6a:4c:bc:ef:cd:86:47:df:62:90:18:1f:18:
```

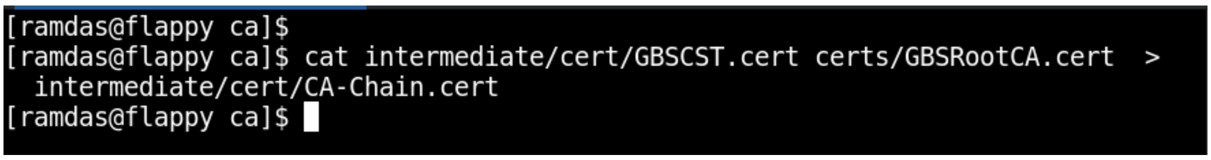
We can clearly see, the issuer is **GBS Root Certifying Authority** and the subject is **GBS CST Certifying Authority**. This is not a self-signed certificate.

We can also verify this digital certificate by using the Root CA's certificate. We can use the **openssl verify** command for this purpose.



```
[ramdas@flappy ca]$ openssl verify -CAfile certs/GBSRootCA.cert intermediate/cer
t/GBSCST.cert
intermediate/cert/GBSCST.cert: OK
[ramdas@flappy ca]$ 
```

To verify digital certificates issued by Intermediate CA to end user we will need a file containing both the certificate of Intermediate CA and the Root CA. Let's create that file. This is necessary as all the digital certificates in the chain are required to verify end users digital certificate. **Please remember to use the filename GBSCSTCA.cert instead of GBSCST.cert in the command below.**



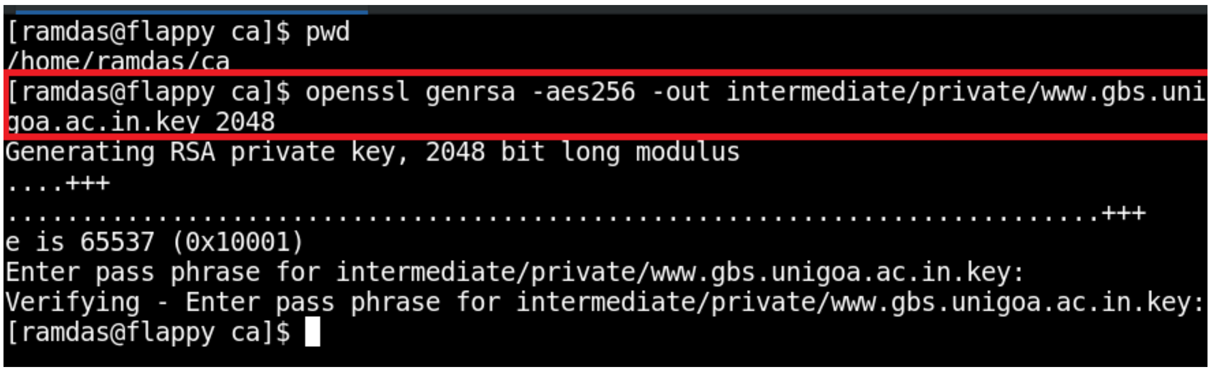
```
[ramdas@flappy ca]$
[ramdas@flappy ca]$ cat intermediate/cert/GBSCST.cert certs/GBSRootCA.cert  >
  intermediate/cert/CA-Chain.cert
[ramdas@flappy ca]$ 
```

CA-Chain.cert now contains both the root and intermediate CA certificates. We have finished setting up our root and intermediate certifying authorities. Now we can issue certificates to end users. The end user could either be an individual or a web server.

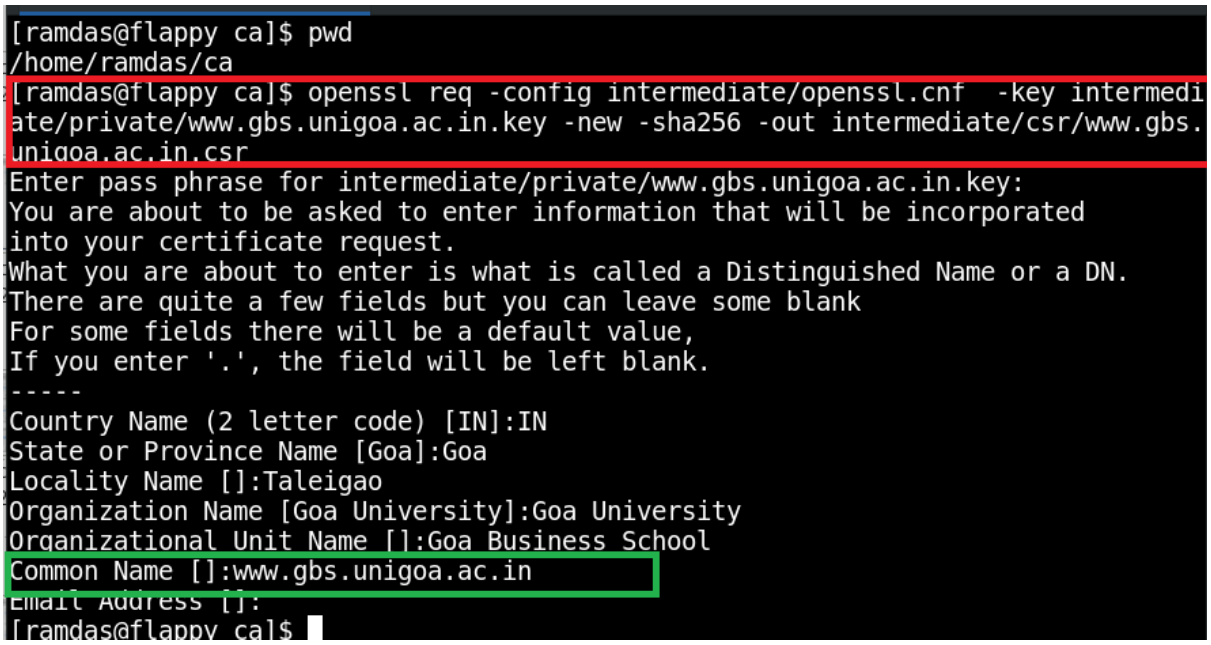
Issuing certificates to end user

We will now create RSA public-private key pairs for web server hosted at **www.gbs.unigoa.ac.in** and for users **Alice, Bob and Darth**.

We will begin by creating public-private key pair for webserver **www.gbs.unigoa.ac.in**, generate a CSR and get it signed by Intermediate CA. We will user **server_cert section** of the configuration file for generating the certificate. This allows the use of certificate for verifying signatures and encrypting session key.

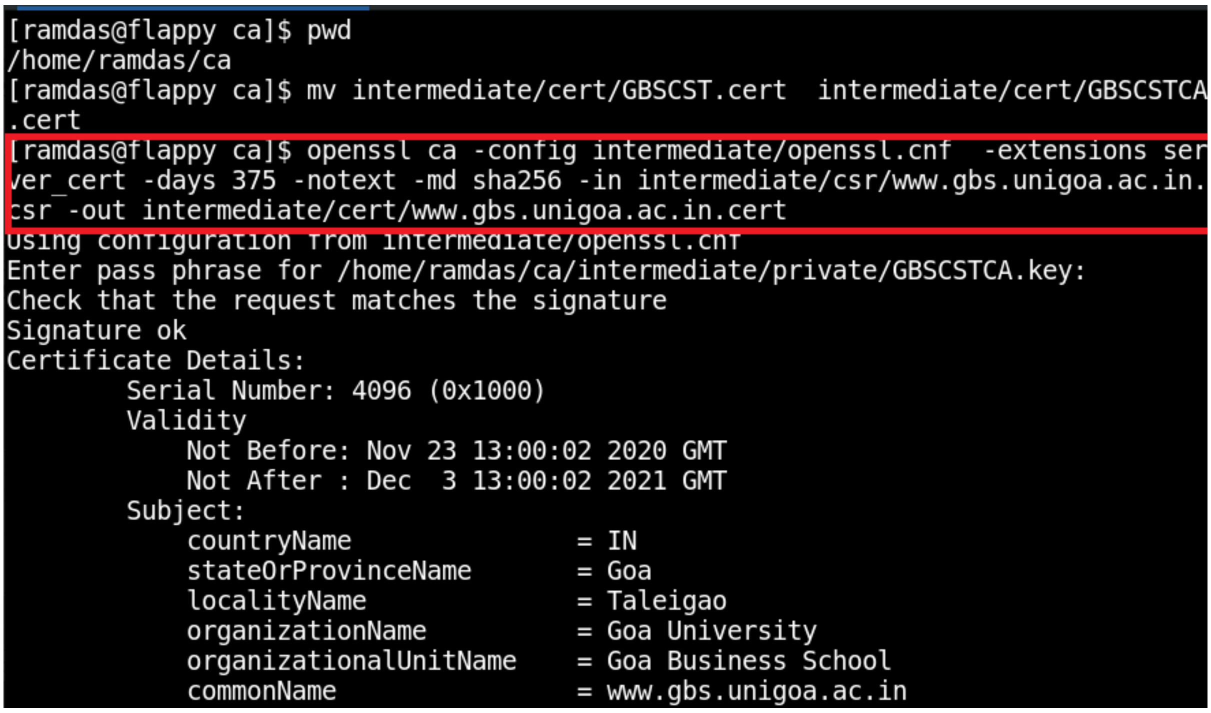


```
[ramdas@flappy ca]$ pwd
/home/ramdas/ca
[ramdas@flappy ca]$ openssl genrsa -aes256 -out intermediate/private/www.gbs.uni
goa.ac.in.key 2048
Generating RSA private key, 2048 bit long modulus
....+++
...........................................................................+++
e is 65537 (0x10001)
Enter pass phrase for intermediate/private/www.gbs.unigoa.ac.in.key:
Verifying - Enter pass phrase for intermediate/private/www.gbs.unigoa.ac.in.key:
[ramdas@flappy ca]$ 
```

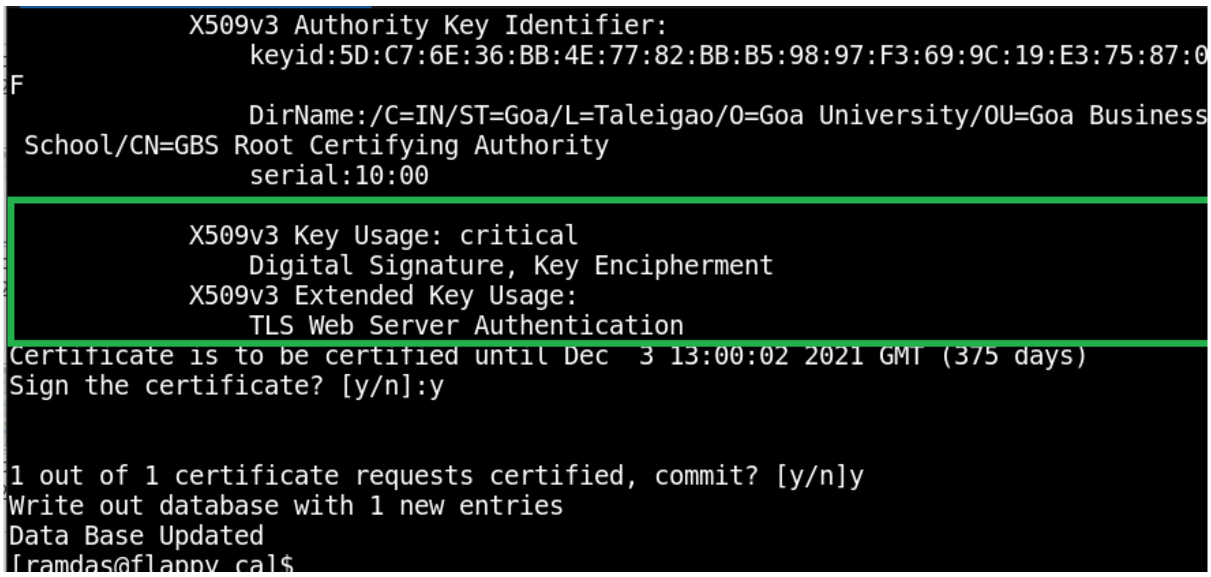


```
[ramdas@flappy ca]$ pwd
/home/ramdas/ca
[ramdas@flappy ca]$ openssl req -config intermediate/openssl.cnf  -key intermedi
ate/private/www.gbs.unigoa.ac.in.key -new -sha256 -out intermediate/csr/www.gbs.
unigoa.ac.in.csr
Enter pass phrase for intermediate/private/www.gbs.unigoa.ac.in.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [IN]:IN
State or Province Name [Goa]:Goa
Locality Name []:Taleigao
Organization Name [Goa University]:Goa University
Organizational Unit Name []:Goa Business School
Common Name []:www.gbs.unigoa.ac.in
Email Address []:
[ramdas@flappy ca]$ 
```

If not done so far, please move **intermediate/cert/GBSCST.cert** to **intermediate/cert/GBSCSTCA.cert**. This is to correct naming mistake we made while generating the Intermediate CA certificate. It needs to be same as what was specified in **intermediate/openssl.cnf** file. Otherwise **openssl ca** command will give 'File not found' error.



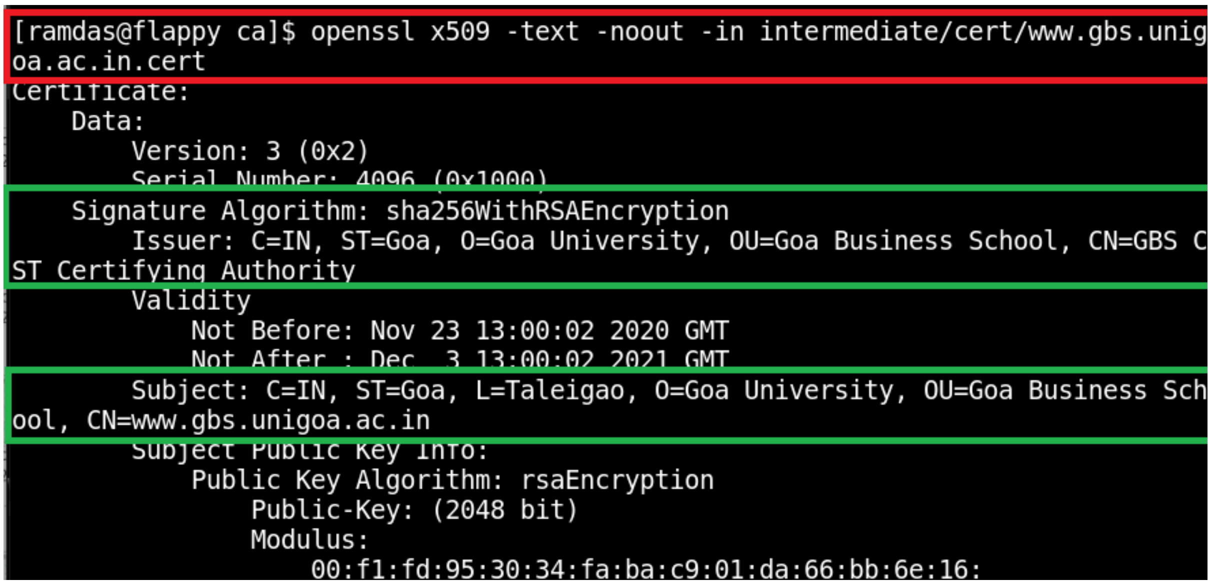
```
[ramdas@flappy ca]$ pwd
/home/ramdas/ca
[ramdas@flappy ca]$ mv intermediate/cert/GBSCST.cert  intermediate/cert/GBSCSTCA
.cert
[ramdas@flappy ca]$ openssl ca -config intermediate/openssl.cnf  -extensions ser
ver_cert -days 375 -notext -md sha256 -in intermediate/csr/www.gbs.unigoa.ac.in.
csr -out intermediate/cert/www.gbs.unigoa.ac.in.cert
Using configuration from intermediate/openssl.cnf
Enter pass phrase for /home/ramdas/ca/intermediate/private/GBSCSTCA.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4096 (0x1000)
        Validity
            Not Before: Nov 23 13:00:02 2020 GMT
            Not After : Dec  3 13:00:02 2021 GMT
        Subject:
            countryName               = IN
            stateOrProvinceName       = Goa
            localityName              = Taleigao
            organizationName          = Goa University
            organizationalUnitName    = Goa Business School
            commonName                = www.gbs.unigoa.ac.in
```



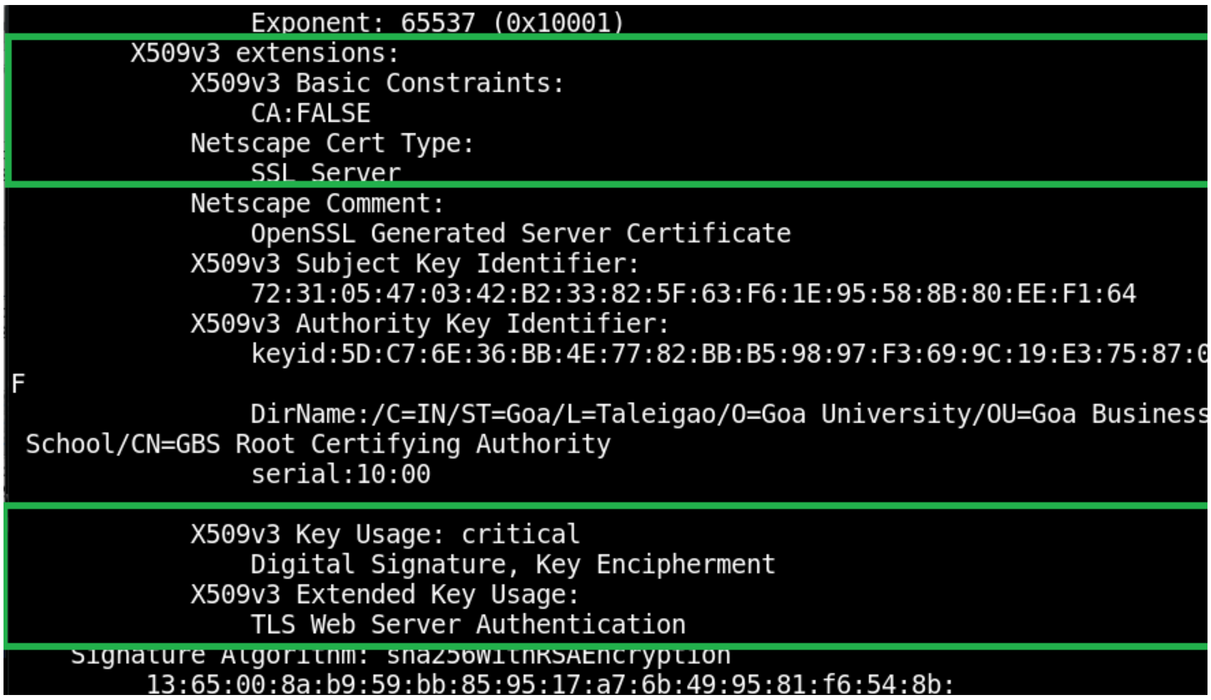
```
            X509v3 Authority Key Identifier:
                keyid:5D:C7:6E:36:BB:4E:77:82:BB:B5:98:97:F3:69:9C:19:E3:75:87:0
F

                DirName:/C=IN/ST=Goa/L=Taleigao/O=Goa University/OU=Goa Business
 School/CN=GBS Root Certifying Authority
                serial:10:00

            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
Certificate is to be certified until Dec  3 13:00:02 2021 GMT (375 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
[ramdas@flappy ca]$
```

We can be seen above the digital certificate can be used for verifying digital signatures and encrypting session keys.

We can also check contents of the certificate.

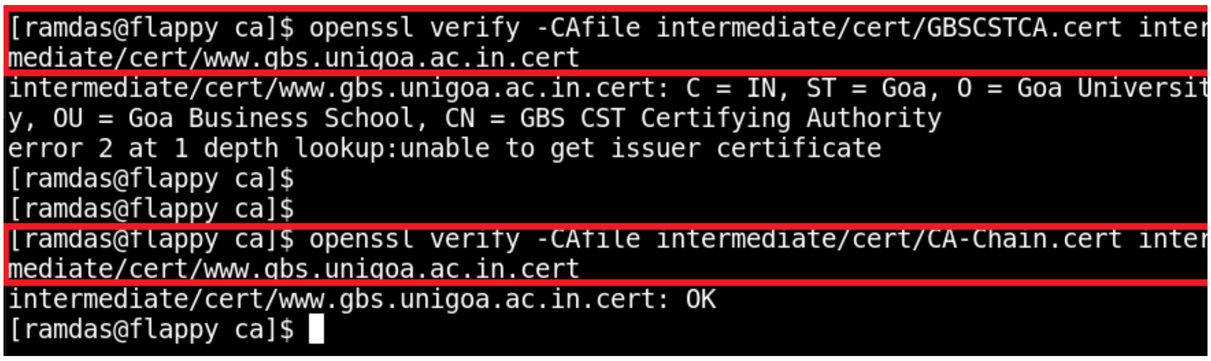


```
[ramdas@flappy ca]$ openssl x509 -text -noout -in intermediate/cert/www.gbs.unig
oa.ac.in.cert
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4096 (0x1000)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=IN, ST=Goa, O=Goa University, OU=Goa Business School, CN=GBS C
ST Certifying Authority
        Validity
            Not Before: Nov 23 13:00:02 2020 GMT
            Not After : Dec  3 13:00:02 2021 GMT
        Subject: C=IN, ST=Goa, L=Taleigao, O=Goa University, OU=Goa Business Sch
ool, CN=www.gbs.unigoa.ac.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:f1:fd:95:30:34:fa:ba:c9:01:da:66:bb:6e:16:
```



```
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Cert Type:
                SSL Server
            Netscape Comment:
                OpenSSL Generated Server Certificate
            X509v3 Subject Key Identifier:
                72:31:05:47:03:42:B2:33:82:5F:63:F6:1E:95:58:8B:80:EE:F1:64
            X509v3 Authority Key Identifier:
                keyid:5D:C7:6E:36:BB:4E:77:82:BB:B5:98:97:F3:69:9C:19:E3:75:87:0
F
                DirName:/C=IN/ST=Goa/L=Taleigao/O=Goa University/OU=Goa Business
 School/CN=GBS Root Certifying Authority
                serial:10:00

            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
    Signature Algorithm: sha256WithRSAEncryption
         13:65:00:8a:b9:59:bb:85:95:17:a7:6b:49:95:81:f6:54:8b:
```

We can also verify the certificate.

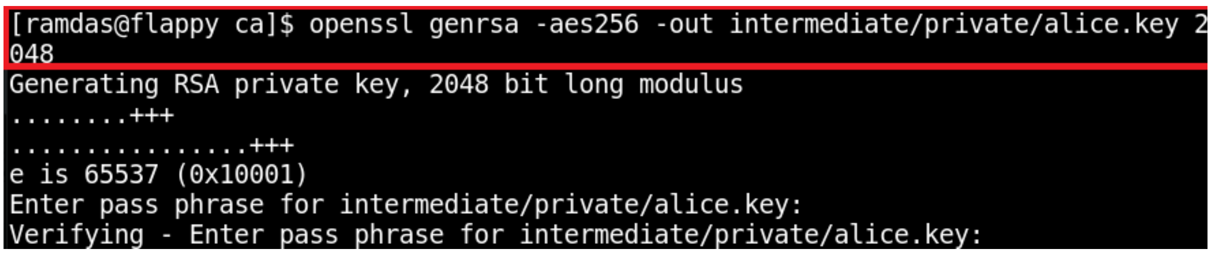


```
[ramdas@flappy ca]$ openssl verify -CAfile intermediate/cert/GBSCSTCA.cert inter
mediate/cert/www.gbs.unigoa.ac.in.cert
intermediate/cert/www.gbs.unigoa.ac.in.cert: C = IN, ST = Goa, O = Goa Universit
y, OU = Goa Business School, CN = GBS CST Certifying Authority
error 2 at 1 depth lookup:unable to get issuer certificate
[ramdas@flappy ca]$
[ramdas@flappy ca]$
[ramdas@flappy ca]$ openssl verify -CAfile intermediate/cert/CA-Chain.cert inter
mediate/cert/www.gbs.unigoa.ac.in.cert
intermediate/cert/www.gbs.unigoa.ac.in.cert: OK
[ramdas@flappy ca]$
```

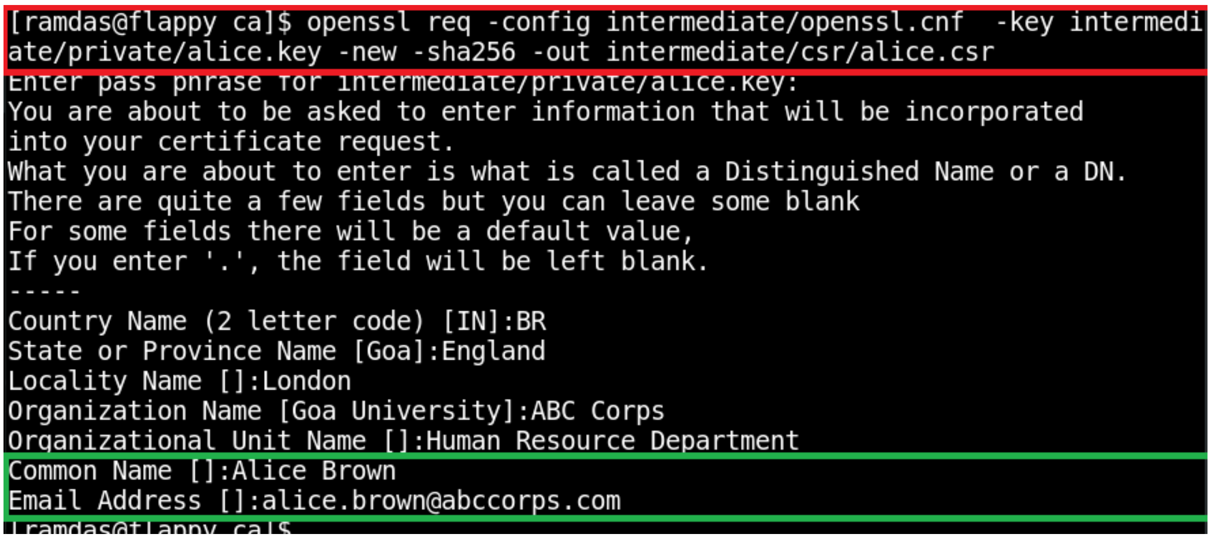
The first command tries to verify the certificate using only the certificate of Intermediate CA. This fails as the public key of Intermediate CA cannot be verified due to missing Root CA certificate. But when we use **CA-Chain.cert** file, the verification succeeds as all the certificate in the chain till the self-signed root certificate is available for performing verification.

Under the normal circumstances the end user will create the public-private key pair and only the CSR will we sent to the Intermediate CA for signature to generate digital certificate. The generated digital certificate will be issued to the end user.

Similarly we can create the certificates for Alice, Bob and Darth. Specify the name of the person when prompted for Common Name during CSR generation. Don't forget to give some dummy email address when prompted for email address. Use your imagination to give other information like country code, state, locality, etc.

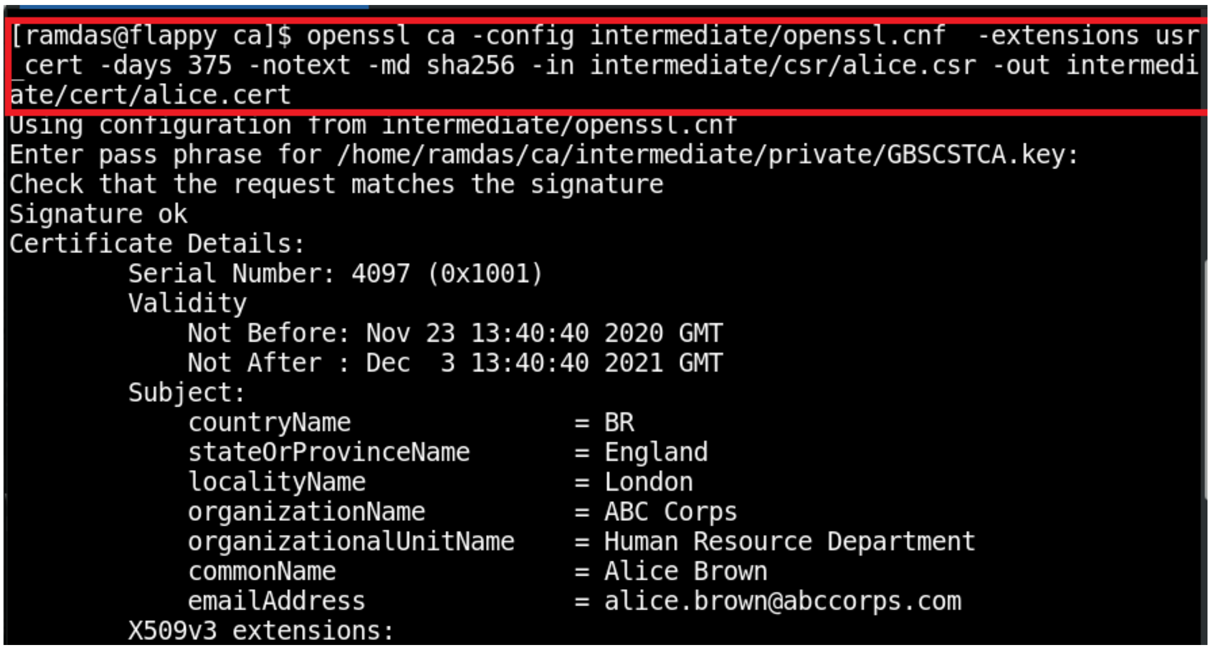


```
[ramdas@flappy ca]$ openssl genrsa -aes256 -out intermediate/private/alice.key 2
048
Generating RSA private key, 2048 bit long modulus
........+++
................+++
e is 65537 (0x10001)
Enter pass phrase for intermediate/private/alice.key:
Verifying - Enter pass phrase for intermediate/private/alice.key:
```

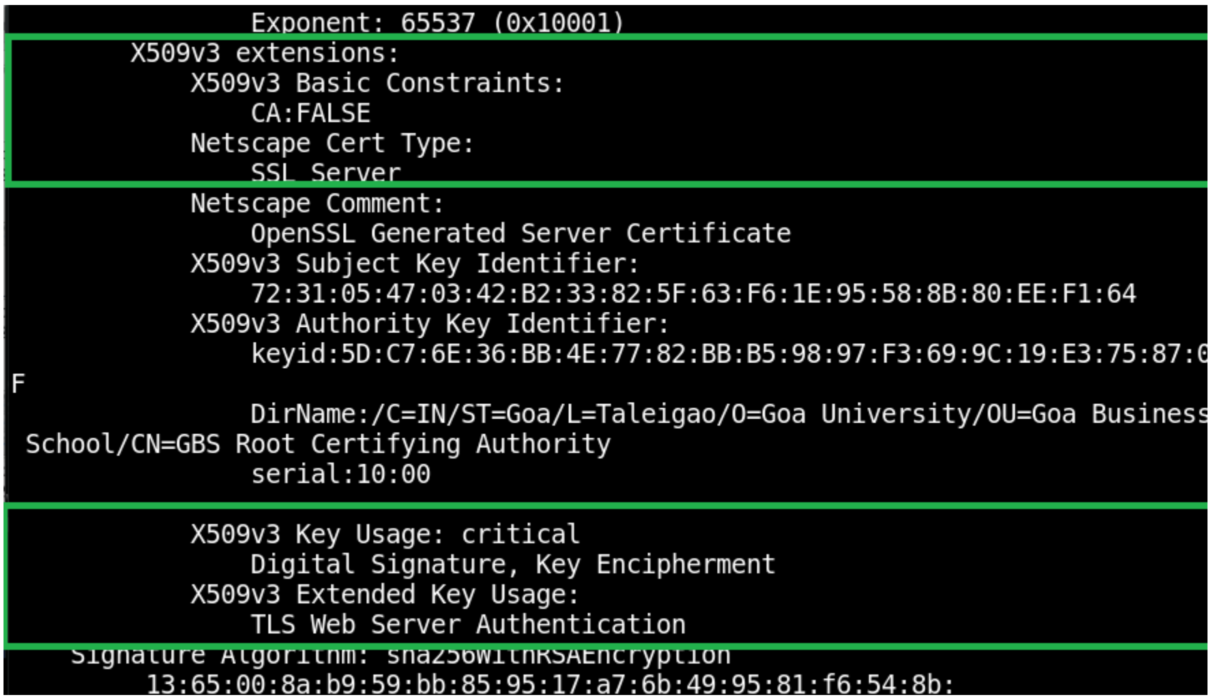


```
[ramdas@flappy ca]$ openssl req -config intermediate/openssl.cnf  -key intermedi
ate/private/alice.key -new -sha256 -out intermediate/csr/alice.csr
Enter pass phrase for intermediate/private/alice.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [IN]:BR
State or Province Name [Goa]:England
Locality Name []:London
Organization Name [Goa University]:ABC Corps
Organizational Unit Name []:Human Resource Department
Common Name []:Alice Brown
Email Address []:alice.brown@abccorps.com
[ramdas@flappy ca]$
```

Important: When generating the user certificate use the **usr_cert section** of **intermediate/openssl.cnf** file. (**option -extensions usr_cert**).



```
[ramdas@flappy ca]$ openssl ca -config intermediate/openssl.cnf  -extensions usr
_cert -days 375 -notext -md sha256 -in intermediate/csr/alice.csr -out intermedi
ate/cert/alice.cert
Using configuration from intermediate/openssl.cnf
Enter pass phrase for /home/ramdas/ca/intermediate/private/GBSCSTCA.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4097 (0x1001)
        Validity
            Not Before: Nov 23 13:40:40 2020 GMT
            Not After : Dec  3 13:40:40 2021 GMT
        Subject:
            countryName               = BR
            stateOrProvinceName       = England
            localityName              = London
            organizationName          = ABC Corps
            organizationalUnitName    = Human Resource Department
            commonName                = Alice Brown
            emailAddress              = alice.brown@abccorps.com
        X509v3 extensions:
```

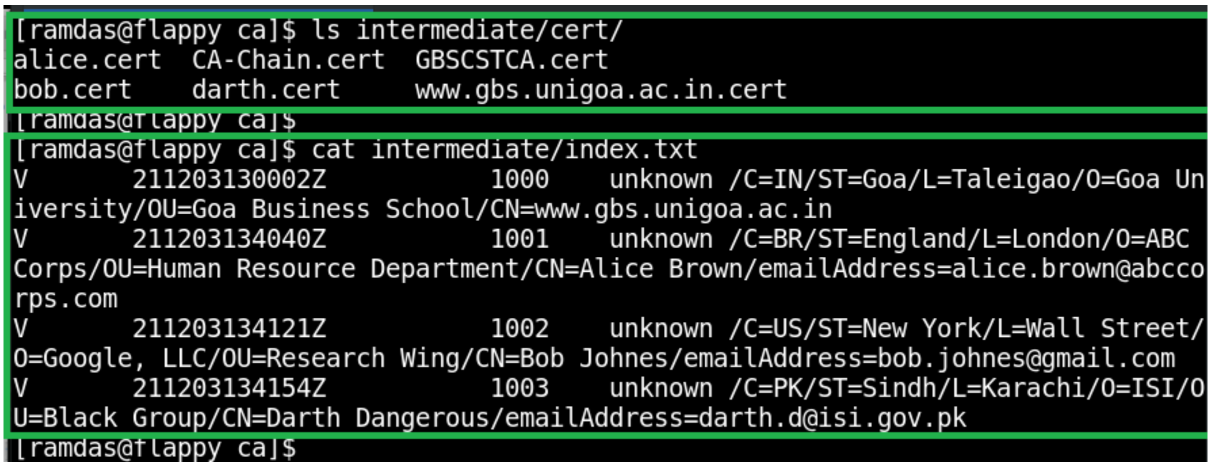


```
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Cert Type:
                SSL Server
            Netscape Comment:
                OpenSSL Generated Server Certificate
            X509v3 Subject Key Identifier:
                72:31:05:47:03:42:B2:33:82:5F:63:F6:1E:95:58:8B:80:EE:F1:64
            X509v3 Authority Key Identifier:
                keyid:5D:C7:6E:36:BB:4E:77:82:BB:B5:98:97:F3:69:9C:19:E3:75:87:0
F
                DirName:/C=IN/ST=Goa/L=Taleigao/O=Goa University/OU=Goa Business
 School/CN=GBS Root Certifying Authority
                serial:10:00

            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
    Signature Algorithm: sha256WithRSAEncryption
         13:65:00:8a:b9:59:bb:85:95:17:a7:6b:49:95:81:f6:54:8b:
```

The keys should be stored in **intermediate/private subdirectory**, CSRs **in intermediate/csr subdirectory** and the digital certificates in **intermediate/cert** subdirectory.

The file **index.txt** in **intermediate subdirectory** will contain information about all the certificates issued by Intermediate CA.

In a similar manner, create the digital certificates for Bob and Darth.

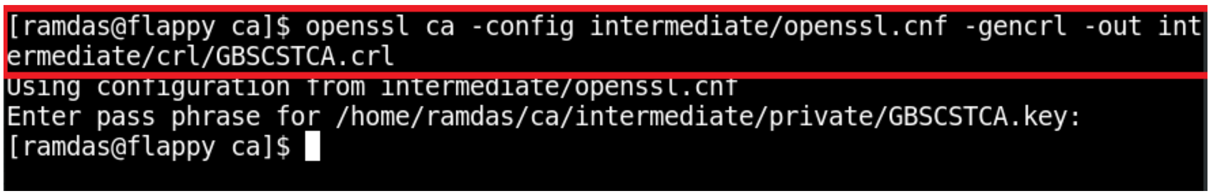


```
[ramdas@flappy ca]$ ls intermediate/cert/
alice.cert  CA-Chain.cert  GBSCSTCA.cert
bob.cert    darth.cert     www.gbs.unigoa.ac.in.cert
[ramdas@flappy ca]$
[ramdas@flappy ca]$ cat intermediate/index.txt
V       211203130002Z           1000    unknown /C=IN/ST=Goa/L=Taleigao/O=Goa Un
iversity/OU=Goa Business School/CN=www.gbs.unigoa.ac.in
V       211203134040Z           1001    unknown /C=BR/ST=England/L=London/O=ABC
Corps/OU=Human Resource Department/CN=Alice Brown/emailAddress=alice.brown@abcco
rps.com
V       211203134121Z           1002    unknown /C=US/ST=New York/L=Wall Street/
O=Google, LLC/OU=Research Wing/CN=Bob Johnes/emailAddress=bob.johnes@gmail.com
V       211203134154Z           1003    unknown /C=PK/ST=Sindh/L=Karachi/O=ISI/O
U=Black Group/CN=Darth Dangerous/emailAddress=darth.d@isi.gov.pk
[ramdas@flappy ca]$
```

Revoking certificates

CA maintains a Certificate Revocation List (CRL) which provides a list of certificates that have been revoked. A client application, such as a web browser, can use a CRL to check authenticity of server's digital certificate. A server application, such as Apache, can use a CRL to deny access to clients that are no longer trusted (if the client is also authenticated using digital certificate). TLS/SSL protocol required by HTTPS does support such mutual authentication using digital certificates. In either case they will need to know the where the CRL is located. A CA can store the URL of the CRL file in the certificate by using **crlDistributionPoints** extension.

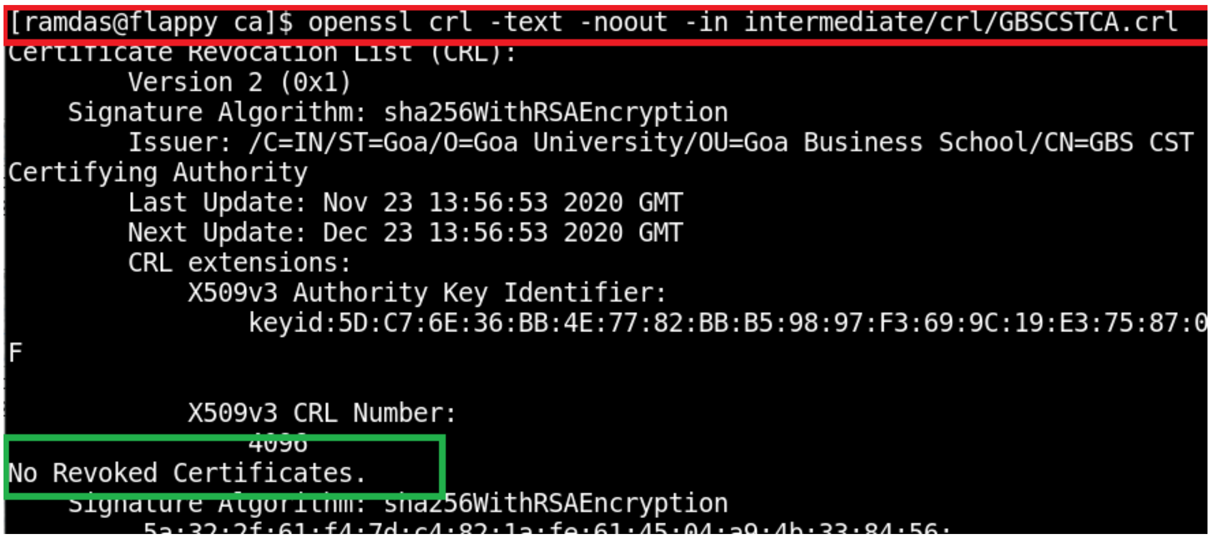
Our Intermediate CA needs to create a CRL file to store revocation information. We will call the file **GBSCSTCA.crl** and it will be stored in **intermediate/crl subdirectory**. We use **openssl ca** command for this purpose.



```
[ramdas@flappy ca]$ openssl ca -config intermediate/openssl.cnf -gencrl -out int
ermediate/crl/GBSCSTCA.crl
Using configuration from intermediate/openssl.cnf
Enter pass phrase for /home/ramdas/ca/intermediate/private/GBSCSTCA.key:
[ramdas@flappy ca]$ 
```

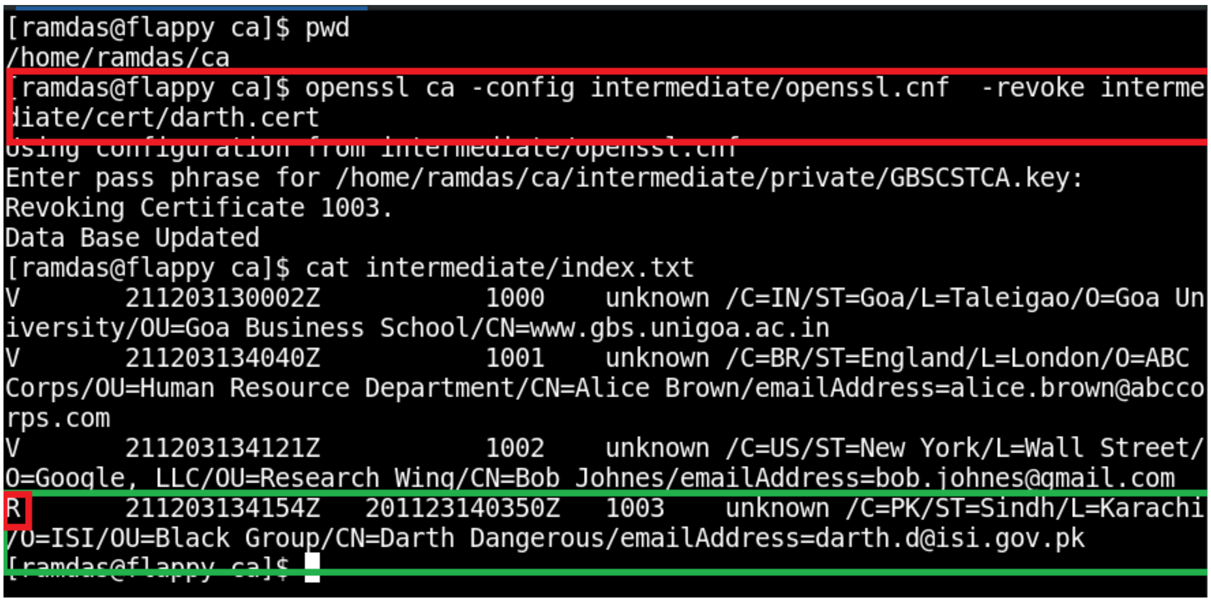
The information stored in CRL is protected against unauthorised modification by signing it with Intermediate CA private key. Hence we will have to supply the password to decrypt the private key.

We can look at the contents of the CRL file that we have created. It will right now contain no revoked certificates. We use **openssl crl** command for this purpose. It will report 'No Revoked Certificate'.



```
[ramdas@flappy ca]$ openssl crl -text -noout -in intermediate/crl/GBSCSTCA.crl
Certificate Revocation List (CRL):
        Version 2 (0x1)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: /C=IN/ST=Goa/O=Goa University/OU=Goa Business School/CN=GBS CST
Certifying Authority
        Last Update: Nov 23 13:56:53 2020 GMT
        Next Update: Dec 23 13:56:53 2020 GMT
        CRL extensions:
            X509v3 Authority Key Identifier:
                keyid:5D:C7:6E:36:BB:4E:77:82:BB:B5:98:97:F3:69:9C:19:E3:75:87:0
F

            X509v3 CRL Number:
                4096
No Revoked Certificates.
    Signature Algorithm: sha256WithRSAEncryption
         5a:32:2f:61:f4:7d:c4:82:1a:fe:61:45:04:a9:4b:33:84:56:
```

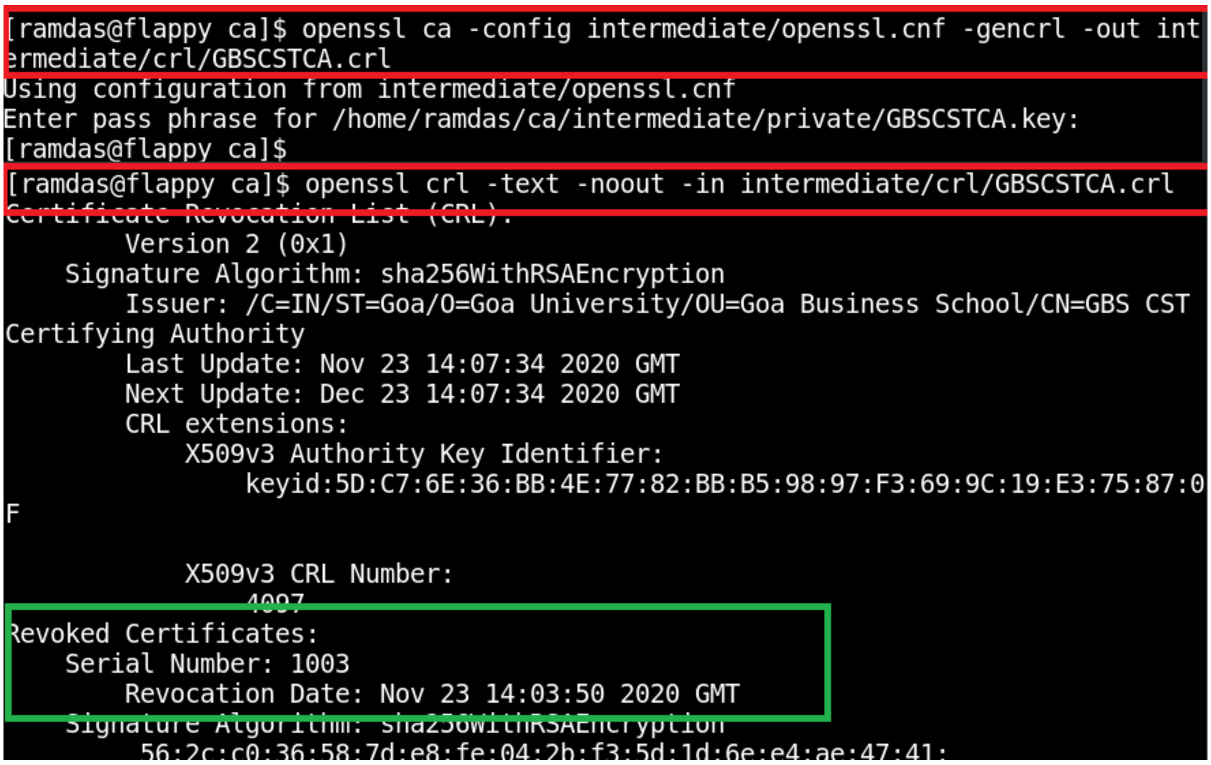
To revoke a certificate we can use **openssl ca** command. We will use it to revoke the certificate of user Darth.



```
[ramdas@flappy ca]$ pwd
/home/ramdas/ca
[ramdas@flappy ca]$ openssl ca -config intermediate/openssl.cnf  -revoke interme
diate/cert/darth.cert
Using configuration from intermediate/openssl.cnf
Enter pass phrase for /home/ramdas/ca/intermediate/private/GBSCSTCA.key:
Revoking Certificate 1003.
Data Base Updated
[ramdas@flappy ca]$ cat intermediate/index.txt
V       211203130002Z           1000    unknown /C=IN/ST=Goa/L=Taleigao/O=Goa Un
iversity/OU=Goa Business School/CN=www.gbs.unigoa.ac.in
V       211203134040Z           1001    unknown /C=BR/ST=England/L=London/O=ABC
Corps/OU=Human Resource Department/CN=Alice Brown/emailAddress=alice.brown@abcco
rps.com
V       211203134121Z           1002    unknown /C=US/ST=New York/L=Wall Street/
O=Google, LLC/OU=Research Wing/CN=Bob Johnes/emailAddress=bob.johnes@gmail.com
R       211203134154Z   201123140350Z   1003    unknown /C=PK/ST=Sindh/L=Karachi
/O=ISI/OU=Black Group/CN=Darth Dangerous/emailAddress=darth.d@isi.gov.pk
[ramdas@flappy ca]$
```

After the command is run, you can look at the contents of **index.txt** file. All the other certificates are marked as valid (V), But Dart's certificate is marked as revoked (R).

The change has only happened in the **index.txt** file. The CRL file has not been updated. So every time we revoke a certificate the CRL need to be recreated. After recreating the CRL if we now look at its contents, we will see that the Darth's certificate has been added to the CRL. The certificate with serial number 1003 is the Darth's certificate.



```
[ramdas@flappy ca]$ openssl ca -config intermediate/openssl.cnf -gencrl -out int
ermediate/crl/GBSCSTCA.crl
Using configuration from intermediate/openssl.cnf
Enter pass phrase for /home/ramdas/ca/intermediate/private/GBSCSTCA.key:
[ramdas@flappy ca]$
[ramdas@flappy ca]$ openssl crl -text -noout -in intermediate/crl/GBSCSTCA.crl
Certificate Revocation List (CRL):
        Version 2 (0x1)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: /C=IN/ST=Goa/O=Goa University/OU=Goa Business School/CN=GBS CST
Certifying Authority
        Last Update: Nov 23 14:07:34 2020 GMT
        Next Update: Dec 23 14:07:34 2020 GMT
        CRL extensions:
            X509v3 Authority Key Identifier:
                keyid:5D:C7:6E:36:BB:4E:77:82:BB:B5:98:97:F3:69:9C:19:E3:75:87:0
F

            X509v3 CRL Number:
                4097
Revoked Certificates:
    Serial Number: 1003
        Revocation Date: Nov 23 14:03:50 2020 GMT
    Signature Algorithm: sha256WithRSAEncryption
         56:2c:c0:36:58:7d:e8:fe:04:2b:f3:5d:1d:6e:e4:ae:47:41:
```

The tricky part about using CRL is that CA is required to regularly update the CRL file as and when the certificates are revoked. Anyone who wishes to use the CRL to check status of the certificates also needs to download and maintain a local copy of the CRL file.

A better solution would be to allow online status check of the certificates. We can use Online Certificate Status Protocol (OCSP) for this purpose.

Online Certificate Status Protocol

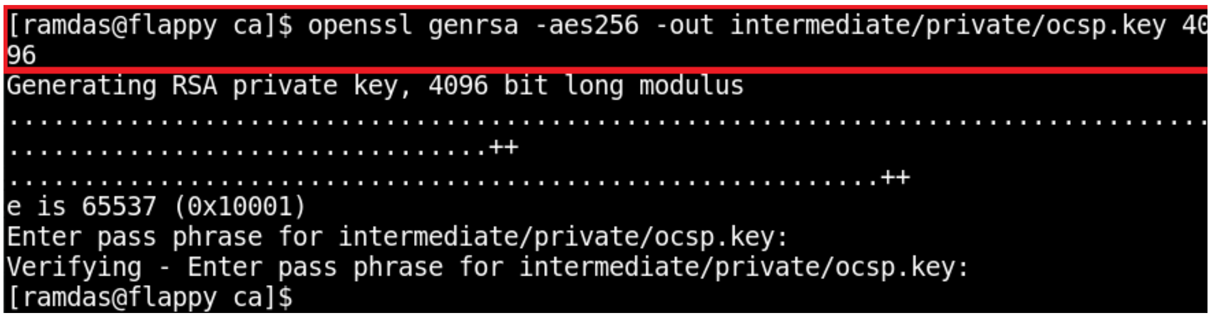
The Online Certificate Status Protocol (OCSP) was created as an alternative to Certificate Revocation Lists (CRLs). Similar to CRLs, OCSP enables a requesting party (eg, a web browser) to determine the revocation status of a certificate.

When a CA signs a certificate, it can typically include an OCSP server address in the certificate. This is similar in function to **crlDistributionPoints** used for CRLs. We can use **authorityInfoAccess** extension to point to the address where OCSP server is hosted.

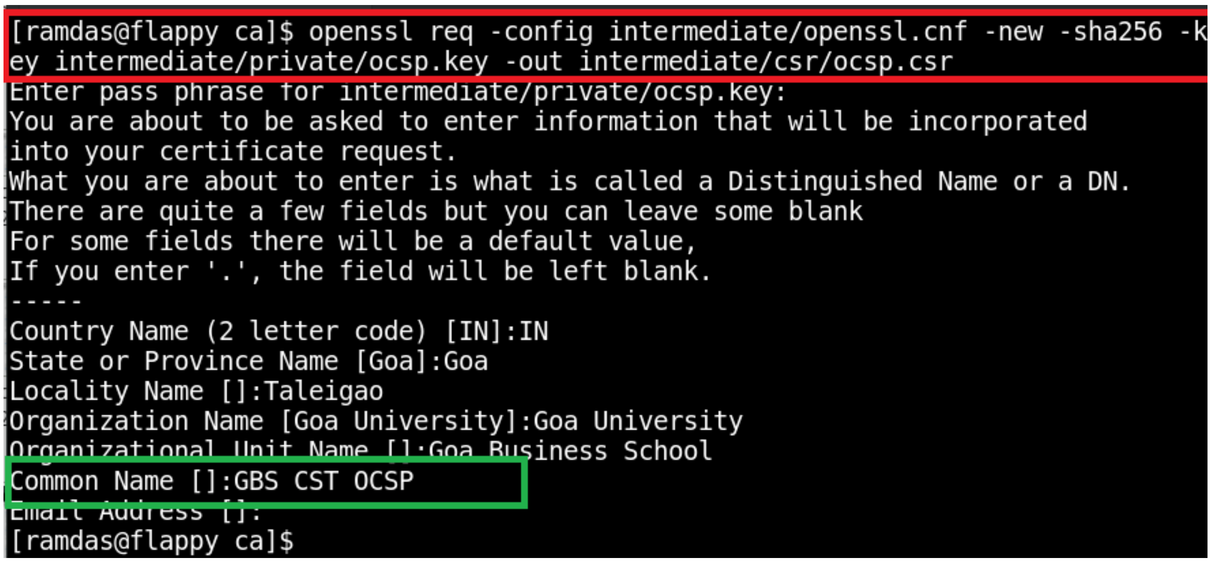
Here, when a web browser is presented with a server certificate, it will send a query to the OCSP server address specified in the certificate. At this address, an OCSP responder listens to queries and responds with the revocation status of the certificate.

For the purpose of security, all information exchanged with the OCSP server needs to be authenticated. Server will sign all the information its sends to the requesting party to protect it against unauthorised modification. For this purpose the OCSP server creates its own public-private key pair and gets a digital certificate issued that is made available to anyone who wishes to use the service.

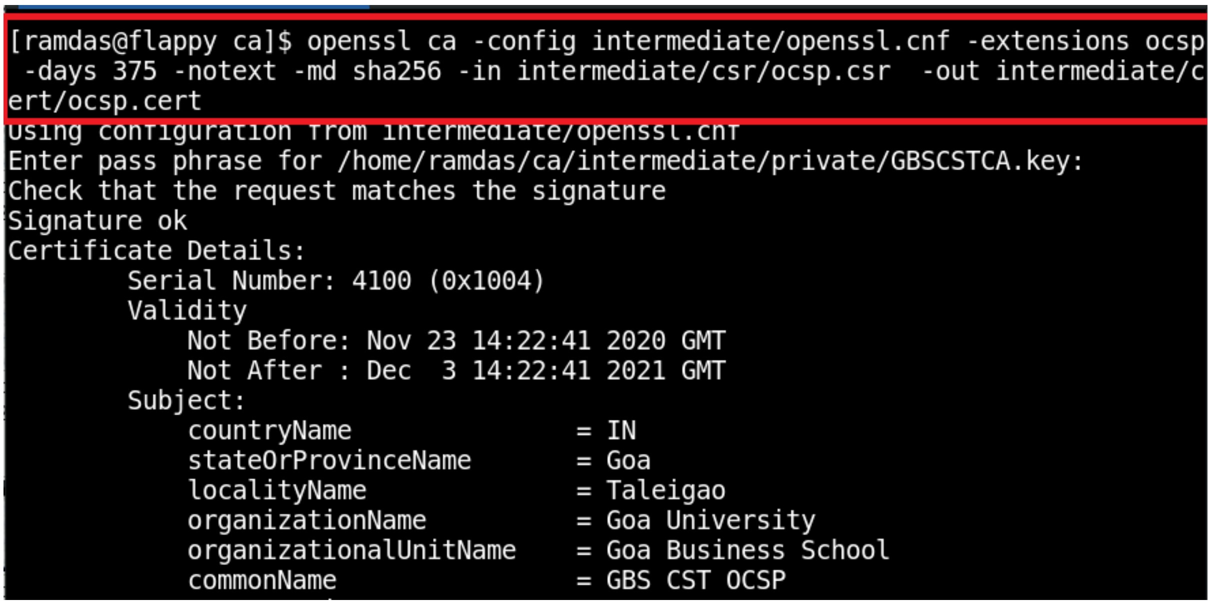
In our case since the OCSP server autheticates the certificate issued by Intermediate CA, the certificate of OCSP server will be signed by Intermediate CA.



```
[ramdas@flappy ca]$ openssl genrsa -aes256 -out intermediate/private/ocsp.key 40
96
Generating RSA private key, 4096 bit long modulus
.............................................................................
................................++
..........................................................++
e is 65537 (0x10001)
Enter pass phrase for intermediate/private/ocsp.key:
Verifying - Enter pass phrase for intermediate/private/ocsp.key:
[ramdas@flappy ca]$
```



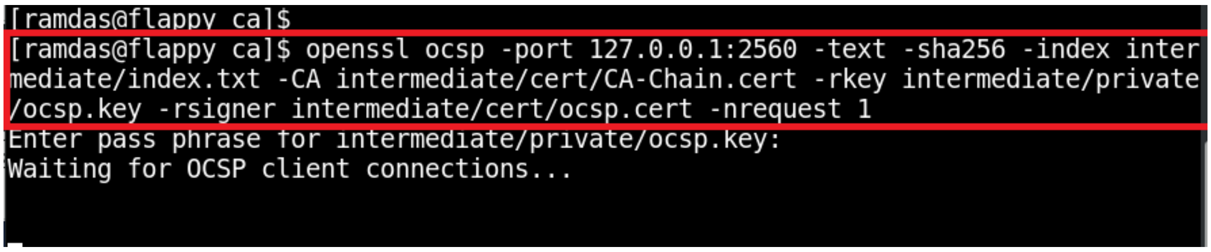
```
[ramdas@flappy ca]$ openssl req -config intermediate/openssl.cnf -new -sha256 -k
ey intermediate/private/ocsp.key -out intermediate/csr/ocsp.csr
Enter pass phrase for intermediate/private/ocsp.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [IN]:IN
State or Province Name [Goa]:Goa
Locality Name []:Taleigao
Organization Name [Goa University]:Goa University
Organizational Unit Name []:Goa Business School
Common Name []:GBS CST OCSP
Email Address []:
[ramdas@flappy ca]$
```



```
[ramdas@flappy ca]$ openssl ca -config intermediate/openssl.cnf -extensions ocsp
 -days 375 -notext -md sha256 -in intermediate/csr/ocsp.csr  -out intermediate/c
ert/ocsp.cert
Using configuration from intermediate/openssl.cnf
Enter pass phrase for /home/ramdas/ca/intermediate/private/GBSCSTCA.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4100 (0x1004)
        Validity
            Not Before: Nov 23 14:22:41 2020 GMT
            Not After : Dec  3 14:22:41 2021 GMT
        Subject:
            countryName               = IN
            stateOrProvinceName       = Goa
            localityName              = Taleigao
            organizationName          = Goa University
            organizationalUnitName    = Goa Business School
            commonName                = GBS CST OCSP
```

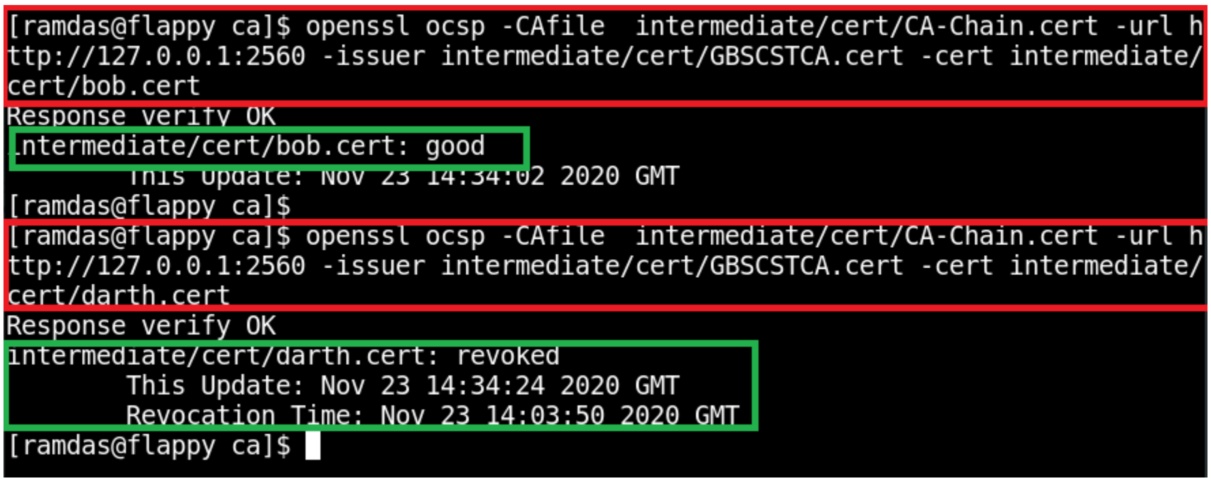
Once the certificate is created now we can start the OCSP server. We can **use openssl ocsp** command for this purpose. We will invoke it with **–port option** which will specify the port and the IP address where the server will listen for incoming requests. In our case the IP address is local IP address and the port number is 2560.

We will also need to pass the private key of OCSP server **(-rkey option**), the digital certificate (-**rsigner option**) and the certificates of both the CAs (-**CA option**). The **–nrequest option** specifies number of requests that will be processed. If you plan to send more than one request, please use a value higher than 1, say 10.



```
[ramdas@flappy ca]$
[ramdas@flappy ca]$ openssl ocsp -port 127.0.0.1:2560 -text -sha256 -index inter
mediate/index.txt -CA intermediate/cert/CA-Chain.cert -rkey intermediate/private
/ocsp.key -rsigner intermediate/cert/ocsp.cert -nrequest 1
Enter pass phrase for intermediate/private/ocsp.key:
Waiting for OCSP client connections...
```

Now from another shell invoke the **openssl ocsp** command in client mode. Use **–url** option to specify the server address. The option **–CAfile** is needed to verify the signature of the OCSP response. The certificate whose status is to be checked is passed using **–cert** option. This should be the last option. The issuer of the certificate is passed using **–issuer** option.



```
[ramdas@flappy ca]$ openssl ocsp -CAfile  intermediate/cert/CA-Chain.cert -url h
ttp://127.0.0.1:2560 -issuer intermediate/cert/GBSCSTCA.cert -cert intermediate/
cert/bob.cert
Response verify OK
intermediate/cert/bob.cert: good
        This Update: Nov 23 14:34:02 2020 GMT
[ramdas@flappy ca]$
[ramdas@flappy ca]$ openssl ocsp -CAfile  intermediate/cert/CA-Chain.cert -url h
ttp://127.0.0.1:2560 -issuer intermediate/cert/GBSCSTCA.cert -cert intermediate/
cert/darth.cert
Response verify OK
intermediate/cert/darth.cert: revoked
        This Update: Nov 23 14:34:24 2020 GMT
        Revocation Time: Nov 23 14:03:50 2020 GMT
[ramdas@flappy ca]$
```

As can be seen above, the status of Bob's certificate is returned as good, but the status of Darth's certificate is reported as revoked. Both certificates pass verification as they are both legitimate certificate issued by the Intermediate CA and the signature is genuine. Any browser should both verify the certificate and check its revocation status. Only if both the tests are passed, the certificate can be considered as authentic.

Unlike in the case of CRLs, the OCSP allows us to check the status of certificate online, which is more convenient.